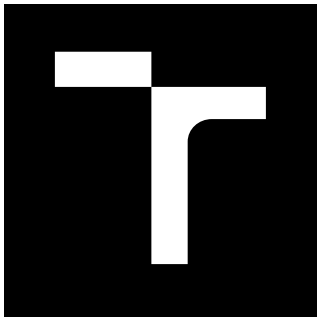

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ
FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

ÚSTAV TELEKOMUNIKACÍ
DEPARTMENT OF TELECOMMUNICATIONS

AUTOMATIZOVANÉ ZPRACOVÁNÍ ZÁZNAMŮ SÍŤOVÝCH SLUŽEB V OS LINUX
AUTOMATED PROCESSING OF NETWORK SERVICE LOGS IN LINUX

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — **Jan Hodermarsky**
AUTHOR

VEDOUCÍ PRÁCE — **Ing. Petr Ilgner**
SUPERVISOR

BRNO 2019

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

Bakalářská práce

bakalářský studijní obor **Informační bezpečnost**
Ústav telekomunikací

Student: Jan Hodermarsky ***ID:*** 185926
Ročník: 3 ***Akademický rok:*** 2018/19

NÁZEV TÉMATU:

Automatizované zpracování záznamů síťových služeb v OS Linux

POKYNY PRO VYPRACOVÁNÍ:

Cílem práce je navrhnout a realizovat systém pro vyhodnocení souborů se záznamy síťových služeb provozovaných na OS Linux (log soubory) v reálném čase. V těchto souborech budou vyhledány pokusy o narušení bezpečnosti serveru dané služby (např. mnoho chybných přihlášení, pokusy o vyčerpání prostředků a další). Na detekované události systém vhodně reaguje, např. zablokováním dané komunikace. Systém bude udržovat databázi metadat detekovaných útoků. Popis veškerých detekovaných útoků aplikace bude konfigurovatelný pomocí textových souborů. V rámci práce budou vytvořeny popisy pro detekci v záznamech služeb Apache HTTP Server, Postfix, Dovecot, OpenSSH.

Vytvořená aplikace bude provozována jako démon OS Linux. Bude mít implementováno jednoduché textové rozhraní umožňující ovládání démona (odblokování komunikace, zobrazení blokovaných spojení, aktivace/deaktivace určité služby). Pro zobrazení informací o detekovaných hrozbách správci bude k dispozici jednoduchá webová služba. Aplikace bude sestavena v jazyce Python, případně C/C++.

DOPORUČENÁ LITERATURA:

[1] NEMETH, E., G. SNYDER, T. HEIN, B. WHALEY a D. MACKIN. UNIX and Linux System Administration Handbook, 5th Edition. 2017. ISBN 9780134278308.

[2] GIFT, Noah a Jeremy M JONES. Python for Unix and Linux system administration. Beijing: O´Reilly, 2008, xix, 433 s. ISBN 978-0-596-51582-9.

Termín zadání: 1.2.2019 ***Termín odevzdání:*** 27.5.2019

Vedoucí práce: Ing. Petr Ilgner
Konzultant:

prof. Ing. Jiří Mišurec, CSc.
předseda oborové rady

Abstrakt

Tato kvalifikační práce se věnuje návrhu a implementaci software pro profylaktickou analýzu journal souborů v reálném čase za účelem následné detekce hrozeb z nich zřejmých. Software se zaměřuje především na síťové služby, resp. jejich journal soubory, a to vše na platformě Linux. V journal souborech jsou vyhledávány pokusy o narušení bezpečnosti serverových služeb na základě definic v konfiguračním souboru. Předkládaná práce si klade za cíl dosáhnout co největší míry versatility pro jednoduchou konfiguraci služeb nových, jež mají být pomocí tohoto software sledovány, potažmo chráněny. Významným přínosem práce je přítomnost webového rozhraní pro správu software snadno a pohodlně nejen lokálně, ale i vzdáleně skrz protokol HTTP.

Abstract

This thesis is focused on design and implementation of software for a prophylactic real-time logfile analysis and a consequent threat detection apparent therein. The software is to concentre particularly on network services, respectively, on the log files thereof, on Linux platform. The log files are observed for potential security breach attempts in regard to respective service as defined in the configuration file. The present thesis purports to reach the largest extent of versatility possible for a straightforward configuration of a new service which is to be monitored and protected by the software. An important asset of the work is a web-based interface accessible through HTTP protocol which allows the software to be administered remotely with ease.

Klíčová slova

analýza logů, analýza souborů journal, detekce anomálie, detekce hrozeb, journal soubory, formát journal souborů, IPS, kontrola journal souborů v reálném čase

Keywords

logfile analysis, journal file analysis, anomaly detection, threat detection, journal file, journal file format, IPS, real–time logfile monitoring, logfile

Bibliografická citace

Hodermarsky, Jan. *Automatizované zpracování záznamů síťových služeb v OS Linux*. Brno, 2019. Bakalářská práce. Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav telekomunikací. 50 s.

Čestné prohlášení

Prohlašuji, že bakalářskou kvalifikační práci na téma „Automatizované zpracování záznamů síťových služeb v OS Linux“ jsem vypracoval sámostatně pod vedením vedoucího práce. Dále prohlašuji, že veškeré prameny a zdroje informací, které byly použity k sepsání a vytvoření této práce, jsou uvedeny v seznamu použité literatury a jiných informačních zdrojů, resp. uvedeny ve formě komentářů v souborech zdrojového kódu. Prohlašuji také, že v souvislosti s vytvořením této práce jsem neoprávněně nezasáhl do práv třetích osob.

Tato práce je vypracována v souladu se směrnicí rektora č. 72/2017 Úprava, odevzdávání a zveřejňování závěrečných prací ve stavu k 27. 5. 2019 a také s vyhláškou Ústavu telekomunikací na FEKT Úprava, odevzdání a hodnocení závěrečných prací jakožto vnitřní normou ve smyslu čl. 16 směrnice.

.....

Jan Hodermarsky
V Brně, 27. 5. 2019

Poděkování

Na tomto místě bych velmi rád poděkoval všem, kteří mi během studia pomáhali a podporovali mě. Jmenovitě bych rád svůj enormní vděk adresoval Ing. Petrovi Ilgnerovi, vedoucímu práce, za podnětné konzultace a nadstandardně vstřícný a pohodový přístup po celou dobu, během níž tato práce vznikala. Rovněž chci poděkovat svým rodičům za konstantní nejen psychickou podporu během celého studia.

Obsah

Strana

1. Úvod

S explosivním růstem Internetu a technologií se fenomén informační bezpečnosti stává stále více významným, a to nejen pro podniky a velké organizace. Dokonce i organizace či jednotlivci, kteří přímo na svých zařízeních nedisponují daty specifické významnosti pro zachování utajení, představují potenciální terč pro četné druhy útoků, jež mohou mít za následek ohrožení či újmu na statcích[1].

Informační systémy se stávají potupem času stále více komplexními a služby běžící na serverech mnohem více heterogenními. To je dáno kromě jiného nepřeberným množstvým rozličných producentů, vzájemných konkurentů, na trhu software, ať už jde o webové servery, e–mailové servery či servery souborové. Každý takový element, služba běžící na systému, představuje bezpečnostní risiko a povrch pro potenciální kybernetický útok.

Inherentní součástí bezpečnosti a správného fungování informačního systému je garance spolehlivosti, legality a pertinence zpracovávaných transakcí.[11] Toto nezbytně vyžaduje ukládání jistých informací, které konkrétní transakci charakterizují.[11] Není nutné ukládat informace o každé transakci, vše by mělo být přiměřené. Test přiměřenosti by měl být prováděn na základě významnosti a informační hodnoty ukádané informace. Pro zajištění alespoň kontroly výše zmíněné triády se stává důležitým efektivní monitorování systému a rychlé odhalení problému. Informace o transakcích jsou běžně ukádány do souborů – běžně nazývaných journal (počeštěně žurnál) soubory, anglicismem často nazývány rovněž log soubory. V této práci se bude pracovat s pojem journal soubor.

Tato práce se věnuje návrhu a implementaci software pro profylaktickou analýzu journal souborů v reálném čase a následné detekci hrozeb z nich pozorovatelných. Software se má zaměřit především na síťové služby, resp. jejich journal soubory, a to vše na platformě Linux. V journal souborech jsou vyhledávány pokusy o narušení bezpečnosti té které serverové služby, potažmo samotného systému. Jelikož si předkládaná práce klade za cíl především dosažení maximální versatility pro jednoduchou konfiguraci služeb nových, jež mají být pomocí tohoto software sledovány a chráněny, je patrné, že služby, jak plyne ze zadání, tedy Apache HTTP Server, Postfix, Dovecot a OpenSSH nepředstavují v žádném případě taxativní výčet služeb, se kterými bude software schopen pracovat a které bude podporovat. Prostředkem implementace byl zvolen skriptovací jazyk Python. Tato volba byla učiněna na základě myšlenky, že čas programátora je cennější než výpočetní čas počítače, *nota bene* v ohledu na premisu, že výsledné řešení pravděpodobně nebude bez dalšího příliš často nasazováno v systémech s malou výpočetní silou, není tedy nutné příliš lpět na minimalizaci požadavků na výpočetní prostředky.

Druhá kapitola se věnuje vysvětlení podstaty journal souborů, stručně rozebírá jejich typologii, popisuje možné přístupy k jejich struktuře a způsoby jejich analýzy. Kapitola třetí se zabývá v rámci bezpečnostní domény detekcí anomálií v journal souborech. Teoreticky je zde nastíněna podstata detekce nejen statické, kterou řešení této práce využívá, ale i dynamické. Samostatná kapitola je věnována komparativní analýze trhu software, které nabízí obdobnou funkcionalitu a které se

[1] Za statek zde lze považovat například zachování kvality služby či nerušený soukromý život.

zaměřují na stejný segment trhu, jako tomu je v případě této práce. V následující separátní kapitole je rozebrána podstata, funkcionalita, základní stavební kameny a fundamentální principy, základní myšlenky, na nichž je řešení postaveno, a které utváří a razí ducha celé práce, partikulárně ducha části praktické. V kapitole šesté jsou rozebrány vybrané technické aspekty implemetnovaného řešení. Obsah této kapitoly má sloužit primárně snazšímu pochopení zdrojového kódu a vzájemných souvislostí v něm. Následuje kapitola pojednávající o předdefinovaných pravidlech detekce vybraných služeb, které byly vymezeny v zadání práce. Osmá kapitola pojednává o problematice ochrany a zpracování osobních údajů v kontextu software pracujícího se soubory journal, a to zejména v souvislosti se zpracováním osobních údajů fyzických osob. V kapitole deváté zde předkládané práce je krátce nastíněno zamyšlení se nad možnými zlepšeními *pro futuro*, lakonicky se zde kontempluje nad možnými směry, kterými se řešení nad rámec této práce může v budoucnosti vyvíjet a ubírat.

Věřím, že tato práce by mohla být přínosem (nad rámec účelu `per se` – akademická kvalifikace autora) alespoň pro opensource komunitu. Aby zde nenastala situace podobná, jak se tomu stalo v případě software SSHGuard[2], pojmenuji praktické řešení této práce poněkud alternativně, přesto sotva nevýstižně – GoofyGoHome (zkráceně GGH). Název je inspirován jménem postavy vytvořené Walt Disney Company. Domnívám se, že takto extravagantní název není na závadu, jelikož právě takto nápadné názvy přilákají často nejvíce pozornosti širší veřejnosti – a to je žádoucí.

Pro srozumitelnost a aesthetické důvody budu ve zbytku této práce označovat praktické řešení, software GoofyGoHome, pouze řešením. Pod pojmem démon se v této práci rozumí integrální část řešení, která hlídá journal soubory a reaguje na transakce v nich dokumentované. Tento démon rovněž vytváří lokální linux socket jako server (v rámci architektury klient–server) pro komunikaci s klientem a klientem webovým. Slovem webový klient se zde rozumí démon, který figuruje jako HTTP server a poskytuje přístup k ovládání a ke konfiguraci démona skrz protokol HTTP. Klientem CLI se zde rozumí `CLI` rozhraní pro ovládání démona na lokálním zařízení minimalistickým způsobem.

[2]SSHGuard je jedním z konkurenčních software, pro detailnější popis viz sekci 4.7. V době, když software SSHGuard vznikal, podporoval ochranu pouze SSH protokolu a byl schopen sledovat pouze jeho journal soubor, později se však SSHGuard bohatě a široce rozvíjel a dnes svou funkcionalitou předčí mnoho alternativních řešení na trhu. Nešťastným krokem zde však byl onen nevhodně zvolený název, který tomuto produktu zůstal až do dnešního dne, a to navzdory tomu, že tento software v současné době je s to podporovat celou řadu služeb, nikoli toliko SSH, a disponuje nadstandardně bohatou funkcionalitou. Změna názvu produktu představuje nejednoduchý ukol, který aby byl efektivním, je mnohdy i finančně náročným. Na základě délky času, který byl potřebný k dostání se k informaci o existenci tohoto software, je možno učinit závěr, že takto nešťastně zvolený název produktu, jako je tomu v tomto případě, způsobuje nedosažení popularity konkurenčních projektů, ačkoli jsou sotva srovnatelně výkonné či funkcionálně bohaté.

2. Journal soubory

Pro zajištění optimálního stavu bezpečnosti systému v celé jeho šíři je kromě jiného potřeba postarat se o možnost zpětné kontroly transakcí[3]. K tomuto účelu slouží journal soubory. Takřka každá aplikace dnes vytváří journal soubory, do kterých zapisuje informace o svých transakcích (událostech). Mezi zpětnou kontrolu lze podřadit rovněž i kontrolu, která probíhá téměř v reálném čase.

Pomocí dat v journal souborech lze zjistit např. počty přístupů a požadavků na web server, chybové hlášky, ze kterých lze následně zjistit kde a proč nastala chyba, počet odeslaných e–mailů, (ne)úspěšné pokusy o authentizaci a další. Uchovávání dat není *per se* primárním účelem journal souborů. Jak již bylo řečeno v úvodu, účelem je primárně zajištění spolehlivosti, legality a pertinence transakcí.

Journal soubory poskytují informace o sledu událostí, které se v rámci konkrétní aplikace udály. Struktura journal souborů (logů) se mění v závislosti na tom, které službě patří. Existuje zde nicméně několik zaužívaných principů, na které je dobré při vývoji software, který logy produkuje, a vytváří tak bázi dat ve formě journal souborů, dbát. Rozumně vyložená nezávazná pravidla poskytuje např. Splunk[16]. Nejdůležitější se jeví být zásady, jakými jsou využívání struktury klíč – hodnota, snadná čitelnost souborů (ASCII kodování, nikoli binární formát) či využívání časových razítek a unikátních ID pro každou transakci.[16] Je potřeba podotknout, že v praxi tyto prinicpy občas splněny nejsou, případně jsou splněny toliko částečně, a jejich automatizované zpracovávání se tak stává poměrně náročným.

2.1. Základní typy journal souborů

Kritérií, podle kterých by se journal soubory daly dělit, je mnoho. Podle některých autorů lze journal soubory jen stěží dobře rozdělit do více skupin podle způsobu ukládání transakcí.[39] Existuje nicméně několik standardů, které jsou mnohými producenty dodržovány. Dodržování struktur journal souborů však nelze v žádném případně po producentech software vynucovat, a tudíž ani nelze předpokládat jakoukoli všobecnou apriorní uniformitu v této oblasti.

Ve většině journal souborech jsou data o událostech a transakcích (logy) uložena ve struktuře přirozeného jazyka.[39] Přirozený jazyk je obtížné analyzovat, neboť nedisponuje pevně danou strukturou. I přesto existuje několik standardů pro strukturu a formát journal souborů. Pro operační systém Linux jsou nejvíce relevantní formáty NCSA a ELF.

2.1.1. NCSA formáty

NCSA formáty jsou založeny na struktuře poprvé dané NCSA httpd webovým serverem, odtud také plyne tento název. NCSA formáty rozlišujeme tři, a to „Common

[3]Záznamem transakcí lze zde rozumět vše, co ta která služba běžící na operačním systému či operační systém sám provedl, provádí a kdy se tak dělo.

Log Format“, který je nejběžnější a je-li pojednáváno pouze o NCSA formátu, je tím myšleno právě Common Log Format. Další dva standardy se nazývají NCSA Combined Log Format a NCSA Separate. NCSA Combined Log Format je rozšířením Common Log Format, a to tak, že log podle tohoto standardu obsahuje navíc informaci `referrer`, `useragent` a `cookie`.[22] Dále bude pojednáno o klasickém NCSA Common Log Formátu.

Typickým příkladem aplikací, které vytváří své journal soubory s logy podle standardu NCSA jsou webové servery jako např. Apache, ale také i aplikace využívající SIP[4] protokol pro IP telefonii. NCSA formát ukládá informace o klientech přistupujících na službu webového serveru. Pro FTP servery není NCSA formát vhodný.[1] Nevýhodou je, že NCSA standard nezaznamenává v rámci logů celou HTTP hlavičku, ale toliko URI řetězec s methodou. V případě POST methody nejsme schopni z logu zjistit POST parametry a jejich hodnoty. Ačkoli je tento přístup z pohledu analýzy útoků značnou nevýhodou, má své opodstatnění. Obsahem hodnot methody POST jsou často citlivé informace. K journal souborům může teoreticky mít přístup širší okruh subjektů, což je jeden z hlavních důvodů nutnosti absence citlivých informací.

NCSA formát je tzv. pevný formát, což znamená, že podobu ukládaných záznamů nelze upravovat podle potřeby.[1] Obsahuje IP adresu klienta, jméno či ID uživatele činícího požadavek, HTTP methodu, vyžádaný zdrojový soubor (URN), časové razítko, kód odpovědi (status kód) serveru na požadavek a několik dalších informací marginálnějšího významu. Neobsahuje však informace jako `useragent` či `referral`. NCSA formát pracuje s kódováním ASCII.

Z pohledu bezpečnosti se jeví býti zajímavou zejména informace o uživateli, který daný požadavek na server odeslal. V majoritní většině případů nabývá tato položka v logu hodnoty „–“, neboť dnes již spíše výjmečně se využívá HTTP authentizace jako způsob authentizace na WWW. Ukázka logu, který v sobě obsahuje uživatele jméno uživatele, který se authentizoval methodou HTTP Basic, je ve výpisu č. 1.

Výpis № 1 Apache log úspěšné HTTP authentizace

```
::1 - john [20/Nov/2018:00:50:23 +0100] "GET /subfolder/80080/ HTTP
    ↪ /1.1" 200 1759 "-" "Mozilla/5.0 (X11; Fedora; Linux x86_64)
    ↪ AppleWebKit/537.36 (KHTML, like Gecko) Chrome/60.0.3112.113
    ↪ Safari/537.36"
```

Notace `::1` znamená, že požadavek byl učiněn z localhostu a třetí položka – john – je authentizovaný uživatel, který GET požadavek na `/subfolder/80080/` učinil.

2.1.2. Extended Log File Format

Extended Log File formát byl vytvořen v reakci na nedostačující množství informací, které ukládá formát NCSA CLF a jiné proprietární specifikace formátu

[4]Pro Common Log Format pro SIP protokol existuje i samostatná RFC specifikace – RFC 6872. Dostupné na: `https://tools.ietf.org/html/rfc6872`.

logů.[3] Na druhou stranu se objevily i požadavky na ohleduplnější zacházení s osobními údaji a zaznamenávání některých informací bylo nežádoucí. NCSA CLF je však rigidní a dle jeho specifikace nelze jakoukoli položku logu vynechat ani přidat položku novou. Takový přístup se v dnešní době v mnoha případech jeví býti nedostatečně vyhovujícím.

2.2. Analýza journal souborů

Aktivita či transakce *grosso modo* kterékoli aplikace běžící na serveru založeném na OS Linux může být automaticky analyzována – mluvíme zde o tzv. „log file analysis“.[22] Analýza může být provedena v mezích rozsahu událostí, které aplikace do journal souboru ukládá. Záludnost této činnosti leží ve variabilním charakteru logů. Logem pro účely této práce se rozumí záznam jedné transakce v rámci jedné aplikace (je však nutno podotknout, že mnohdy jedna transakce vygeneruje vícero logů).

Záznamy v journal souborech mohou mít formu klíč-hodnota. Takový způsob reprezentace dat je poněkud přívětivý, neboť podle názvu klíče lze snadno určit sémantiku hodnoty a lze si tak představit, jaké jsou její vlastnosti a jakých hodnot může nabývat.[39] V této práci budeme předpokládat, že obsah journal souborů nedodržuje žádný ustálený standard. Opodstatněnost daného jevu lze přičítat principu versatility, na kterém je práce postavena. Více k principu versatility v kapitole 5.1.

Jakmile jsou journal soubory naplněny daty, přichází na řadu jejich analýza, tedy následná interpretace a zpracování za účelem získání informace z těchto dat. Analýza je v našem případě nikoli manuální, ale automatizovaná. Řešení této práce se zaměřuje především na služby, které jsou spjaty se sítí, tedy poskytují určitou službu (často neznámému okruhu adresátů) především směrem do sítě Internet. Podpora jiných typů služeb a programů tímto však zůstává nedotknuta.

2.2.1. Journal soubory aplikací

Některé aplikace vytváří, jak je zmíněno výše, své journal soubory ve formě standardizované, jiné zase nikoli. Řešení poskytuje několik předkonfigurovaných pravidel detekce, které mohou v mnoha případech odhalit nekalé počínání uživatele služby.

V kontextu webového serveru Apache lze uvažovat hned o několika typických nežádoucích jednáních uživatelů služby, na které je vhodné patřičně reagovat. V první řadě se zcela bezpochybně jedná o HTTP authentizaci. Log zobrazující úspěšnou HTTP authentizaci lze vidět ve výpisu č. 1. Log neuspěšné HTTP authentizace je vyobrazen ve výpisu č. 2.

Výpis № 2 Apache log neuspěšné HTTP authentizace

```
::1 - goofy [11/Dec/2018:04:21:37 +0100] "GET /lucentom/80080 HTTP
    ↪ /1.1" 401 381 "-" "Mozilla/5.0 (X11; Fedora; Linux x86_64)
    ↪ AppleWebKit/537.36 (KHTML, like Gecko) Chrome/60.0.3112.113
    ↪ Safari/537.36"
```

Uživateli goofy byla authentizace zamítnuta. Důvod zamítnutí NCSA log formát neobsahuje, není tedy zjistitelné, zdali bylo zadáno nesprávné heslo či zda uživatel `goofy` např. neexistuje vůbec. Pro analýzu je zde ztěžejní položka HTTP status kódu (401). Regulární výraz, který nalezne odpovídající HTTP status kód poté může být, jak je uvedeno ve výpisu č. 3.

Výpis № 3 Regulární výraz pro vyhledání HTTP status kodu v Apache logu

```
(GET|POST|OPTIONS) (?P<url>.*) HTTP/[012].[012]" (?P<statuscode
    ↪ >[1-5]\d\d)
```

Status kód je obsažen v symbolickém jménu skupiny `statuscode`. Opakující se status kódy jiné než 2xx mohou indikovat rozličné praktiky útočníků. Útočník se takto může např. pomocí automatizovaných nástrojů využívající techniky hrubé síly snažit najít soubory či adresáře na serveru, které nejsou standardně nikde uvedené a nejsou apriori určeny k nahlížení veřejnosti. To může vést v konečném důsledku až k přístupu k datům a informacím, které mohou býti kritické pro bezpečnost celé systémové infrastruktury.[5]

Systém pro přepravu elektronické pošty Postfix je taktéž častým terčem útoků. Postfix je Mail Transfer Agent (dále jen MTA), jehož ukolem je zpracovávaní a předávání zpráv elektronické pošty mezi servery a lokálně v rámci systému. To znamená, že nezajišťuje komunikaci pomocí protokolů POP ani IMAP. Jako MTA Postfix přijímá a odesílá zprávy elektronické pošty přes síť skrz protokol SMTP. V případě místního dodání může lokální agent Postfixu ukládat zprávy do lokálního úložiště zpráv nebo je předávat specializovanému programu pro doručení pošty – e-mailovému klientu.

Na server SMTP probíhají útoky rovněž velmi často. Primárním zájmem útočníků je získání přístupu k důvěryhodným SMTP serverům velkého měřítka, které jsou posléze zneužívány k masové distribuci spamu. Po SMTP serverech je bohatá poptávka rovněž na černém trhu. Je zbytečné se zde zabývat problematikou detekce spamových zpráv, neboť k tomuto účelu je dostupná řada úzce specializovaných nástrojů. Řešení není příliš vhodné na takto specifickou a specializovanou oblast. Zajímavou se zde opět jeví problematika útoků hrubou silou na authentizační koncové body SMTP serveru.

Výpis № 4 Neuspěšný pokus o authentizaci k SMTP serveru

```
Dec 9 10:57:32 server3 postfix/smtpd[21368]: warning: unknown
    ↪ [185.234.219.56]: SASL LOGIN authentication failed:
    ↪ authentication failure
Dec 9 10:57:32 server3 postfix/smtpd[21368]: lost connection after
    ↪ AUTH from unknown[185.234.219.56]
Dec 9 10:57:32 server3 postfix/smtpd[21368]: disconnect from unknown
    ↪ [185.234.219.56]
```

[5] Často lze takto najít zálohované konfigurační soubory k php aplikacím obsahující authentizační údaje k MySQL databázi nebo hardkódované administrátorské údaje pro přístup do správy webu. Výjimkou není ani situace, kdy si administrátoři ukádají archivované zálohy celého webu do pracovního adresáře Apache (tzv. DocumentRoot).

Ve výpisu č. 4 se subjekt za IP adresou 185.234.219.56 pokusil neúspěšně o authentizaci.

Služby Dovecot a OpenSSH mají jednu významnou společnou charakteristiku. Obě tyto služby využívají ve svých aktuálních verzích v defaultním nastavení tzv. `journald` pro záznam transakcí. Nevytváří si tedy svůj separátní journal soubor, ale využívají centralizovanou službu pro agregaci logů různých služeb `journald`. Technika analýzy logů v journald je hlouběji rozebrána v kapitole č. 6.6.

2.3. Rotační princip

Velikost logovacího souboru roste s časem, a to často takřka přímo uměrně, pokud lze předpokládat přibližně rovnoměrné vytížení služby v čase. Data obsažená v journal souborech po určitém čase ztrácí na relevanci pro účely jejich bezprostřední dostupnosti, popř. jejich ukládání vůbec. Aby se journal soubory nestávaly pořád objemnější až do neunosné míry, existuje zde tzv. rotační princip. Rotační princip je automatizovaný proces, jehož podstatou je archivování journal souborů po určitém čase, kdy informace v něm obsažené již s velkou pravděpodobností nejsou relevantní pro účely nutnosti bezprostřední přístupnosti k takovým datům.

V Linuxu je rotační princip nejčastěji zajišťován příkazem `logrotate`. Neaktuální journal soubory jsou často přesunuty do jiné lokace, mohou být zkomprimovány či poslány e–mailem administrátorovi. Na systémech FreeBSD a MacOS je funkcionalita rotačního principu součástí aplikace `newsyslog`.

`Logrotate` v defaultní konfiguraci zajišťuje archivaci journal souborů služeb, jejichž konfigurace archivování se standardně nachází v jednotlivých konfiguračních souborech v adresáři `/etc/logrotate.d/`. Pro přidání své vlastní služby k logrotate je potřeba vytvořit konfigurační soubor ve zmíněném adresáři.

V rámci rotačního principu se při rozsáhlejší architektuře často využívá institutu tzv. loghostu (logserveru). Loghost je centralizované místo, kam se ukládají archivované journal soubory z ostatních zařízení v síti.

2.3.1. Data retention

V souvislosti s archivováním a rotačním principem je důležité neopomenout, že ukládání journal souborů, přesněji informací v nich se nacházejících, je pro určité subjekty právní povinností *ex lege*. Nejen toto může být vnímáno jako jeden z důvodů pro existenci principu rotace. „Data retention“, jak se tato povinnost uchovávat provozních a lokalizačních údajů v hovorové řeči nazývá, je v České republice v současné době zakotvena v § 97 odst. 3 zákona 127/2005 Sb., o elektronických komunikacích. V souladu s tímto ustanovením je osoba zajišťující veřejnou komunikační síť nebo poskytující veřejně dostupnou službu elektronických komunikací povinna uchovávat po dobu 6 měsíců provozní a lokalizační údaje, které jsou vytvářeny nebo zpracovávány při zajišťování jejích veřejných komunikačních sítí a při poskytovávání jejích veřejně dostupných služeb elektronických komunikací. Osobou zajišťující veřejnou komunikační síť nebo poskytující veřejně dostupnou službu elektronických komunikací se rozumí zejména telefonní operátoři, poskyto-

vatele kabelové televize a poskytovatelé veřejně přístupného připojení zpravidla za úplatu, více viz § 2 písm. n) zmíněného zákona. Rozsah dat, které je nutno uchovávat je upřesněn ve vyhlášce č. 357/2012 Sb., o uchovávání, předávání a likvidaci provozních a lokalizačních údajů.

Tuto povinnost se snažila harmonizovat v rámci uzemí Evropské unie směrnice Evropského parlamentu a Rady 2006/24/ES. V této směrnici šlo primárně o určení jednotné doby pro povinné uchovávání metadat poskytovateli služeb. Tato směrnice tedy stanovuje (stanovovala) povinnost uchovávání jistých dat poskytovateli služeb, přičemž se jedná o přenosy dat nejen Internetem, ale i pevnou telefonní linkou, mobilní sítí, emailovými prostředky či data telefonie přes IP. Důvodem pro toto uchovávání provozních údajů je ulehčení stíhání trestné činnosti, ať už internetové či jiné. K této směrnici se váže jedna zajímavá kuriozita. Jako obvykle, mnoho členských států nestihlo tuto směrnici implementovat včas. Po tom, co německý ústavní soud[26] prohlásil ustanovení této směrnice za protiústavní, z čehož poté plynula neimplementace této směrnice v Německu, bylo proti Německu zahájeno řízení pro porušení SFEU. Než však toto řízení stihlo skončit, prohlásil Evropský soudní dvůr tuto směrnici na návrh Irska[33] v dubnu 2014 za neplatnou.[10] V České republice byla implementace „data retention“ směrnice projednávána taktéž před Ústavním soudem[25], nicméně povinnost uchovávat provozní a lokalizační údaje zůstala, neboť Ministerstvo vnitra připravilo nové znění zákona podle nálezu Ústavního soudu. V Německu se v roce 2015 povinné uchovávání provozních a lokalizačních údajů znovu zavedlo.[10] Nejen v České republice zůstává problematika „data retention“ stále kontroversní. Dne 22. 5. 2019 rozhodoval Ústavní soud opět o ústavnosti současné právní úpravy (Pl. ÚS 45/17). Tentokrát se Ústavní soud rozhodl, že současnou úpravu „data retention“ podrží.

3. Bezpečnostní doména

Útoky na aplikace zvenčí jsou kromě jiného často založené na automatizovaném zjišťování přítomnosti běžících aplikací v systému, které obsahují zranitelnost, která často bývá i veřejně známá. Notorietně známé jsou pro tento typ útoků aplikace webové – např. Wordpress či Joomla. Často se však jedná i o zranitelnosti v nejrůznějších SSH či FTP serverech. Podstatou těchto útoků nejčastěji bývá vložení škodlivého kódu do aplikace, jehož spuštění – které vůbec nemělo správně nastat – způsobí neauthorizovaný průnik do serveru.[22]

Tyto útoky lze detekovat na základě logů v journal souborech jen někdy. Záleží na tom, zdali využití zranitelnosti v aplikaci způsobí její pád. Pokud aplikace nečekaně terminuje, nelze očekávat, že stihne zapsat log o transakci, která pád způsobila, do svého journal souboru. Nicméně, i postdetekce takovýchto útoků se vyplatí, ačkoli log o transakci vznikne sotva v polovině všech případů. Je totiž důležité zabránit stejným útokům do budoucna. Navíc lze na základě jednoho logu, který obsahuje IP adresu útočníka, zablokovat tuto IP adresu ve firewallu, a přerušit tak útočníkovi např. jeho `reverse shell` relaci. Ačkoli to zkušeného útočníka neodradí a najde si cestu kolem, cílem je, jak již bylo řečeno, maximalizace počtu útočníků, kteří budou odrazeni od dalšího postupování v útoku.

Pro detekci známých útoků lze použít regulárních výrazů či komplexnějších method, často spojených s dynamickou detekcí.[22] Aplikace pracují zpravidla na sedmé, nejvyšší, vrstvě ISO/OSI modelu. Řešení je založeno na detekci útoků na základě analýzy journal souborů, jejichž záznamy (logy) jsou vytvářeny právě těmito aplikacemi. Není to však bezvýjimečným pravidlem. Logy jsou sice utvářeny na aplikační vrstvě, mohou však být konstruovány také na základě dat vstvy nižší.

Bezpečnostní politika je soubor pravidel majících za cíl chránit informační systém. Pro zajištění ochrany mají tato pravidla být aplikována koherentním a neredundantním způsobem.[2] Firewall (běžně používané anglické označení, ve frankofonní literatuře se lze setkat s označením pare-feu), je systém umožňující prosazování politiky, tedy pravidel bezpečnosti sítě. Kontroluje a sleduje aplikace a tok dat v systému. Základní typologie rozděluje firewally na stavové a bezstavové.[20] Většina firewallů běžně dostupných na systémech založených na systému UNIX pracují s paradigmatem firewallů stavových i bezestavových. Na transportní a síťové vrstvě jsou činné především firewally, které pracují s pakety a datagramy a přiřazují je známým typům služeb. Takové firewally jsou schopny detekovat nejrůznější anomálie v protokolech. Pro detekci útoků na vrstvě aplikační však nejsou vhodné, protože nejsou s to vidět souvislosti z pohledu vyšší vrstvy.

3.1. Detekce anomálií

Často se objevují logy, záznamy o transakcích (událostech), které se jeví býti odlišnými od očekávaných a předpokládaných záznamů generovaných na základě interakce se službou majoritní části uživatelů. Obvykle se různí od zbytků elementů v datové množině. Jsou tedy pokládány za anomálie. Jedna z nejdůležitějších úloh analýzy journal souborů je jejich detekce a případná reakce na ně.[22]

Detekce anomálií může při větších projektech nabývat znatelné přidané hodnoty i při tzv. vnitřních anomáliích, které nejčastěji znamenají nesprávnou funkčnost programu danou např. nesprávným zpracováním požadavku či jiných vstupních dat.[22] Nejčastějším důvodem užití algoritmů detekce anomálií v souborech journal je však detekce anomálií vnějších. Detekce vnějších anomálií hraje významnou úlohu mimo jiné při detekci útoků na webové aplikace.[22]

Řešení disponuje pouze statickým způsobem detekce útoků. Optimální přístup k detekci je detekce založená na statických pravidlech v kombinaci s detekcí dynamickou.[22]

3.1.1. Statická detekce

Na trhu existují četné IDS (systém detekce intruzí) software založené na statické detekci anomálií, které poskytují ne vždy tu nejspolehlivější[6] ochranu před útoky na služby. Příkladem takového software je `mod_security` modul pro Apache. Fakt, že statická detekce takovýchto software je sotva neprůstřelná, nevede k menší poptávce po takovém software. Zohledněno zde musí být to, že naprostou většinu útočníků je i takový software schopen odradit.

[6] Jedná se o názor autora.

Je potřeba si vždy uvědomit, jaký je prvořadý účel každého přítomného elementu sloužícího ke zvýšení bezpečnosti systému. Každý software by měl být navrhnut tak, aby implicitně nezaváděl do systému žádnou novou zranitelnost. Touto zásadou by se měl řídit i návrh standardů a specifikací.

Webová aplikace by měla být naprogramována tak, aby se přes authentizační formulář byl schopen authentizovat pouze oprávněný subjekt. Pokud webová aplikace není schopna zajistit tuto maximu na sto procent, je to stoprocentní zklamání. Musíme ale akceptovat, že webová aplikace standardně není s to efektivně aplikovat algoritmy pro útoky hrubou silou a pro DoS útoky prakticky vůbec. Proto zde existují samostatná řešení pro každý z těchto povrchů potenciálního útoku. Uveden byl konkrétní příklad, ale tento princip platí universálně. Aplikace musí být bezpečně naprogramovaná, ale mělo by zde kromě toho navíc existovat řešení pro potírání snah o nekalý přístup v rámci zdánlivě validního nakládání s aplikací (např. útok hrubou silou). Dále by mělo být nasazeno i řešení pro zajištění kvality služby, které chrání aplikaci např. před útoky DoS. Gesce prvního a druhého řešení se může zčásti překrývat, což však zpravidla není na závadu. Pokud dva posledně zmíněné druhy software nejsou s to detekovat sto procent útoků, není to stoprocentním zklamáním. Jde totiž o dodatečné sekundární bezpečnostní vrstvy, které slouží k plynulému běhu služby a odrazení potenciálních útočníků. O sekundární, s nadsázkou lze říci že i nadbytečnou, vrstvu se jedná z toho důvodu, že za předpokladu ideálního stavu primárních bezpečnostních elementů – bezpečně naprogramovaná aplikace, dostatečně bezpečná methoda authentizace, atd. – by k aplikaci pravidel na této vrstvě, aniž by došlo k újmě, vůbec nemuselo docházet.

Řešením této práce je software poskytující funkcionalitu, konsistentně s výše uvedeným, pro potírání snah o nekalý přístup v rámci validního nakládání s aplikací. Řešení dbá na to, aby útokům typu bruteforce bylo efektivně zabráněno, avšak rovněž je důležité zachovat transparentnost chráněných služeb pro legitimní uživatele a jejich legitimní transakce. Řešení si tedy neklade za cíl zabránit všem útokům, které se mohou vyskytnout. Prvořadým účelem řešení není pokrýt celý povrch potenciálních útoků na aplikaci, nenahrazuje stoprocentně primární bezpečnostní prkvy zmíněné výše. Slouží toliko ke zvýšení bezpečnosti systému v druhé linii, primárním účelem je rychlé odrazení útočníků, aby nezatěžovali zdroje síťové infrastruktury.

Statické pravidla bezpečnostní politiky se nemění. Jsou založena na definicích známých útoků. V rámci statické analýzy lze rozlišovat dva druhy přístupu k definici bezpečnostní politiky.[29] Negativní bezpečnostní model je založen na tzv. blacklistu. To znamená, že apriorně je veškerá komunikace dovolena a pouze taxativně definované vzory komunikace jsou blokovány.[22] Positivní bezpečnostní model spočívá v apriorním odepření veškeré komunikace, pokud se vzor pro tuto komunikaci nenachází v tzv. whitelistu, tedy seznamu vzorů komunikace validní, která má být propuštěna.

Jelikož je účelem řešení poskytnout sekundární vrstvu bezpečnosti, je uplaňován bezpečnostní model negativní. Pravidla detekce jsou definována v konfiguračním souboru `conf.conf`.

3.1.2. Dynamická dektekce

Je obtížné staticky popsat veškeré typy útoků, neboť neustále vznikají nové methody útoků. Pro zajištění dostatečné bezpečnostní architektury i pro takové útoky, je vhodné zvážit využití mechanismů detekce dynamické.

K dynamické detekci lze přistupovat primárně dvěma způsoby, a to kvalitativně či kvantitativně. Každý z přístupů dává jiné výsledky v rámci klasifikace obsahu journal souborů. Kvantitativní přístup tkví v analýze počtu transakcí a jejich rozložení za jednotku času. Detekce nejčastěji spočívají v počtu přístupů, přenosů či požadavků.[22] Kvalitativní přístup na druhé straně spočívá v detekci mimořádných transakcí na základě jejich vlastností. Nejefektivnější methodologií je využití obou přístupů současně.

Dynamická detekce si zakládá na definování pravidel bezpečnostní politiky „za běhu“, a to nejčastěji na základě strojového učení. Pravidla nejsou definována napřed. Dynamická detekce není součástí zadání práce a přesahuje svou implementační náročností tuto práci. O dynamické detekci dále stručně pojednává podkapitola 9.1 v rámci kapitoly návrhů pro futuro.

4. Přehled existujících řešení

Na trhu v současné době existuje relativně mnoho produktů se stejným účelem a velmi podobnou funkcionalitou, které cílí na totožný segment trhu. Pokusím se zde stručně vyzvednout to nejzajímavější z těch nejpopulárnějších[7] dostupných řešení. Vynechány byly nicméně řešení nesvobodná, tedy všechna distribuovaná pod nesvobodnou[8] licencí. Víceméně všechna níže zmíněná řešení jsou založena na čtení journal souborů, monitorování, zda ta která sledovaná služba je napadána či jinak zneužívána a následné vhodné reakci na takovou událost (nejčastěji zablokování útočící IP adresy ve firewallu).

4.1. Fail2Ban

Software Fail2Ban je jednoznačně předním představitelem řešení v této oblasti. Je naprogramován v jazyce Python. Kód zde není kompilován, nýbrž interpretován Python interpretem. Fail2Ban využívá jazyk Python především za využití jeho objektově orientovaného paradigmatu, jež tento jazyk kromě jiných nabízí. Fail2Ban skenuje journal soubory a blokuje pomocí firewallu IP adresy, které vykazují podezřelé chování. Blokované IP adresy mohou být po určité specifikované době opět odblokovány. Fail2Ban se může pyšnit relativně versatilním přístupem vůči akcím, které lze při detekci hrozby učinit – posílání e-mailu, spuštění jiného programu či zpřísnění nastavení firewallu. Podpora je v defaultní konfiguraci poskytována vícero standardním službám.[4] Fail2Ban obsahuje ve své defaultní konfiguraci spoustu tzv. filtrů, tedy pravidel pro detekci nejrůznějších útoků. Pro spolupráci

[7] Zohlednuje se pouze subjektivní mínění autora.

[8] Ke svobodným licencím více např. MYŠKA, Matěj et al. Veřejné licence v České republice [online]. Brno: Masarykova univerzita, 2014.

s přídavným modulem `mod_security` k webovému serveru Apache je zde speciální pravidlo, jež detekuje v journal souboru Apache události, které `mod_security` detekoval podle svých definic jako škodlivé.[35] Takto sofistikovaných pravidel obsahuje Fail2Ban opravdu mnoho. Tato práce není s to mít ambici předčit funkcionalitu Fail2Ban, která je výsledkem mnoholeté práce profesionálů. Bude se spíše snažit podat na jisté aspekty odlišný pohled, a představit tak funkcionality nové.

4.2. DenyHosts

DenyHosts je další z řady produktů této povahy. Tento software pracuje pouze se službou SSH. Je uskutečněn v jazyce Python, a to s relativně přehlednou strukturou kódu. DenyHosts monitoruje v journal souboru služby SSH neúspěšné pokusy o authentizaci. Filtruje přitom zdrojové IP adresy a kontroluje, jak často se ta která adresa IP pokusila k SSH službě authentizovat. Jakmile je určitý, uživatelem definovaný počet neúspěšných pokusů o přístup překročen, zablokuje DenyHosts danou IP adresu záznamem do souboru `/etc/hosts.deny`. Tento software zaujme především tím, že na WWW stránce[9] producenta je možnost nahlížet centralizovaným způsobem na veškeré zablokované IP adresy, které servery s nasazeným DenyHosts pomocí XML-RPC mechanismu na daný web odesílají. Byť je tato funkcionalita širokou veřejností, především systémovými administrátory, vnímaná jako podstatné plus, je nutno podotknout, že takové počínání je do značné míry v konfliktu s ochranou osobních údajů. Otázce, jak zacházet se zaznamenávanými daty, je věnována samostatná kapitola 8. DenyHosts lze dále vytknout, že podporuje toliko vybrané firewally (`pf` a `iptables`). Ačkoli zadání předkládané práce specifikuje konkrétně Linux OS jakožto cílovou platformu, se kterou má řešení pracovat, snaží se předkládané řešení vyhnout podobným omezením, jak to činí v tomto případě DenyHosts.

4.3. SpyLog

SpyLog je primárně určen pro OS Windows, ale jeho užití v prostředí OS Linux není vyloučeno. SpyLog je uskutečněn v programovacím jazyce Lua. Jedna ze zvláštností jazyka Lua je malá velikost zkompilovaného interpreta Lua skriptů. Lua programy jsou zpravidla nezávislé na platformě, zdrojový kód je za běhu kompilován do bytekodu, který je následně interpretován.[18] Ačkoli je proces kompilace obvykle transparentní a je prováděn za běhu programu, lze jej provést i předem za účelem zvýšení výkonu nebo omezení velikosti obrazu v paměti hostujícího prostředí vynecháním kompilátoru.[18] Předností Lua je ve srovnání s jinými skriptovacími jazyky především jeho rychlost. Lua navíc velmi dobře spolupracuje s volací konvencí jazyka C, takže je velice dobře možné psát část programu v jazyce C a část v Lua. Předností SpyLog je zevrubná atomicita konfiguračních možností pro služby, které mají být chráněny. Řešení této práce se přístupem ke konfiguraci od SpyLog poměrně hojně inspiruje.

[9] Viz `http://stats.denyhosts.net/stats.html`.

4.4. IPBan

Tento ne příliš známý software neohromí žádnými převratnými vlastnostmi. Jedinou výhodou tohoto software je, že je dodáván ve variantách pro Linux, Windows i MacOS. Otázkou je, zdali právě tato danost zaujme. Je naprogramován v jazyce C#, což se projevuje ne zrovna kladně na jeho výkonu. Konfigurační soubory jsou psány v syntaxi značkovacího jazyka XML, jenž však není pro průměrného uživatele tou nejsrozumitelnější a nejpřehlednější volbou.

4.5. Tallow

Předností software Tallow je bezpochybně jeho jednoduchost a nevídaná šetrnost vůči systémovým zdrojům. Je naprogramován v jazyce C, který ho činí velmi efektivním nejen ve smyslu náročnosti na systémové zdroje, ale odpadají zde i veškeré starosti spojené se zastaráváním knihoven a jiných závislostí, což bývá u většiny řešení ve vyšších jazycích a jazycích skriptovacích nezřídka problém. Tallow využívá nativní API journalu systemd. Tallow podporuje pouze protokol SSH, který chrání před opakovanými nezdařenými pokusy o přihlášení. V konfiguračním souboru lze nastavit i položku whitelist, tedy výčet IP adres, na které se pravidla nebudou aplikovat. Nevýhodou je nutnost opětovné kompilace celého programu v případě potřeby úpravy pravidel detekce. Software taktéž spolupracuje pouze s firewallem Netfilter (iptables).

4.6. Pyruse

Pyruse je opensource software licencovaný pod veřejnou licencí GPL 3.0. Jedná se o relativně recentní produkt francouzského původu, který se snaží přesvědčit o své popřednosti svou odlehčeností, přičemž sám autor tuto popřední vlastnost srovnává zejména s konkurenčním Fail2Ban a EpyLog.[8] EpyLog je software, který analyzuje a zpracovává jounal soubory a následně vytváří přehlednou zprávu v HTML, která je nakonec zaslána uživateli na e-mail.[37] Dle slov autora Pyruse jsou Epylog příliš neohebné, a neumožnují tudíž jemnou detailní konfiguraci.[8] Dalším významným popudem pro vytvoření Pyruse bylo taktéž to, že Epylog a Fail2Ban projíždí journal soubory každý samostatně, což představuje mrhání zdroji.[8] Na tomto software je dobře pozorovatelná motivace vytváření nového software (dalšího z řady) tohoto žánru. Předmětem této práce je totiž taktéž konkatenace vícero funkcionalit, zatímco jsou brány v přednost jisté hodnoty subjektivního zájmu.

Konfigurace pravidel detekce je prováděná v JSON notaci. Tato notace je efektivní, má jasná pravidla, ale pro uživatele, který s JSON notací nepřišel již v minulosti do styku, se může jevit dosti matoucí. Nebezpečí zde tkví především v tom, že opomenutí byť jen jediného řídícího znaku, jako je např. čárka či závorka má za následek nemožnost zpracování konfigurace, popř. pád programu, který se takovýto konfigurační soubor pokusí zpracovat. Je opodstatněné předpokládat jistou zručnost administrátora, který má na starost konfiguraci nástroje předmětné povahy, avšak i přesto je vhodné neházet polena pod nohy, kde to není nezbytně nutné.

4.7. SSHGuard

Tento software je dle mého názoru nejvíce propracovaným dílem, pokud jde o svobodná řešení. Je naprogramován v procedurálním strukturovaném jazyce C, který *a priori* není objektově orientován a práce s ním je poměrně více intelektuálně náročná. Skutečnost, že je zhotoven právě v jazyce C, představuje jednu z výhod, kterými tento software ovplývá. Je totiž schopen běžet jako tradiční UNIX démon, a to z důvodu nativity, kterou mu dává jazyk C, velmi efektivně. Je velmi střízlivý, co se týká systémových zdrojů, a je nezávislý na modulech třetích stran, které často v souvislosti s nekompatibilitou verzování činí potíže při interpretaci skriptů např. v jazyce Python. Kromě monitorování journal souborů je SSHGuard schopen pracovat taktéž přímo s protokolem `syslog`.

5. Vybrané devízy implementovaného řešení

Implementované řešení jako inherentní součást této práce je raženo maximou zachování maximální versatility a může se pyšnit několika prvky, které jsou pro něj pregnantní a které jsou u konkurenčních producentů (viz kapitolu 4) postrádány. Některé záležitosti rozebírané v této kapitole jsou pro danou technickou oblast do jisté míry rudimentární, jiné, na druhou stranu, jsou u konkurence v současné době poněkud inovativní a progresivní.

5.1. Versatilita jako prvořadá maxima

Hodnotou, na níž je celé řešení primárně založeno, je versatilita. Leitmotivem je tedy univerzálnost a schopnost software přizpůsobit se nejrůznějším druhům programů (přesněji struktuře jejich journal souborů), jež by administrátor serveru chtěl tímto software monitorovat a chránit. Velký význam je kladen na svobodu uživatele definovat si vlastní pravidla detekce podle vlastních subjektivních, často ojedinělých, požadavků pro konkrétní případy užití. Důvodem, proč řešení této práce staví především na této hodnotě, je nabídnout uživateli lehce konfigurovatelné prostředí pro přizpůsobení exaktní funkcionality dle vlastní potřeby a zamezit pokud možno do největší možné míry ephemérní povaze, která je v současné době nejen pro tento druh software příznačná. Ephemérní povaha v tomto případě spočívá v relativně často se měnící struktuře journal souborů aplikací, což má za následek nutnost změn v algoritmech či jiných částech kódu. Je tedy vyžadována kontinuální údržba software, na kterou nejsou vždy dostatečné prostředky.

5.2. Rozhraní CLI jako základní způsob ovládání démona

Démona lze ovládat ve standardní konfiguraci dvěma způsoby. Lze ho ovládat v rámci rozhraní CLI a dále přes rozhraní webové. Vzhledem k modulární architektuře řešení je možno relativně snadno vytvořit další rozhraní pro komunikaci s démonem.

Rozhraní CLI je šetrné s ohledem na náročnost na hardware, ale i software. Lze s ním pracovat i bez tzv. „Display-Serveru“, jakými jsou např. X11 či Wayland. Klient, jak tato práce toto CLI rozhraní označuje (viz kapitolu 1), pracuje s linux socketem `AF_UNIX` jako prostředkem interprocesní komunikace. Nedisponuje explicitně implementovanou funkcionalitou přenosu komunikace na zařízení odlišná od toho, na kterém je spuštěný démon. Ačkoli by toto nebylo náročné na implementaci, je to nadbytečné, neboť podobného výsledku lze docílit za použití stávající podoby klienta se serverem a klientem SSH, které jsou běžnou součástí výbavy téměř každého serveru.

Démon podporuje řadu příkazů, které mu jsou klienty posílány skrz lokální linux socket. Rozdíl mezi rozhraním CLI a rozhraním webovým spočívá především v adaptaci rozhraní webového pro pohodlnější práci ve formě grafické nadstavby nad příkazy, kterým démon rozumí.

Příkazy zadávané do CLI klienta se potvrzují klávesou Enter. Příkazem `help` lze zobrazit nápovědu s veškerými příkazy, které jsou k dispozici, včetně stručného popisu, k čemu ten který příkaz slouží. Příkazem `version` se zobrazí současná verze démona. Tento příkaz je implementován spíše pro futuro, aby bylo snadné rozpoznat v případě dodatečných úprav programu, jaká verze démona s jakou funkcionalitou je nasazena. Dále následuje řada příkazů pro databázi. Databáze obsahuje tabulky events a eventlog. Databáze a účel těchto tabulek je detailně rozebrán v kapitole 6.2. Obsah každé z tabulek lze zobrazit příkazem `db jménotabulky show [json]`, kde jméno tabulky je buď events nebo eventlog a json je volitelný parametr, který způsobí zobrazení výsledku ve formátu JSON. Tento parametr je přítomen zejména pro vnitřní účely webového rozhraní. Pro odstranění záznamu lze využít příkazu `db eventlog remove ID`, resp. `db events release ID` pro tabulku `events`. ID je v každém případě číslo záznamu v odpovídající tabulce, který má být odstraněn. Příkazu `db events check` možno využít pro manuální kontrolu tabulky `events`, zda je některá z detekovaných událostí v tabulce `events` již „odpykaná“ (tzn. od jejího zápisu do databáze uplynul její `JAILTIME`). Tento příkaz je užitečný zejména při vysokých nastavených hodnotách parametru `DB_EVENT_CHECK_SLEEP` v konfiguračním souboru. Nakonec je zde příkaz `daemon stop`, kterým lze démona zastavit. Démona lze zastavit či restartovat i z webového prostředí, o kterém je pojednáno níže.

5.3. Ovládání démona přes HTTP

Pro aktivaci možnosti ovládat démona skrz web, tedy protokol HTTP, je nutno webový server[10] spustit. Důvodem nutnosti spustit webový server separátně je vyhovění zásadě *privacy by default*, o které je řeč v kapitole 8. Spuštění webového serveru se provede příkazem `./ggh_control_web.py [-d]`, kde `-d` je volitelný argument pro spuštění v režimu debug. Program takto nepřejde do módu démona běžícího na pozadí, ale zůstává navázán v TTY a veškerý výstup je směřován do `stdout`, nikoli do journal souboru, jak je tomu v případě módu démon.

Webový server se spustí bez dalšího na adrese 0.0.0.0 a portu 8008, bude tedy jako služba dostupný na všech rozhraních připojení. Toto chování lze změnit odpovídajícími volitelými parametry `--host` a `--port`. Lze použít i zkráceného formátu zápisu parametrů `-h` resp. `-p`. Pro spuštění serveru tak, aby byl dostupný pouze na určitém rozhraní, je potřeba jako hodnotu parametru `--port` uvést IP adresu žádaného rozhraní. Pokud je webový server spuštěn v módu démona, lze ho ukončit příkazem `./ggh_control_web.py --stop`.

5.3.1. Authentizace webovému rozhraní

Aby se nepovolané osoby nezískaly přístup k informacím dostupným ve webovém rozhraní, je webové rozhraní opatřeno authentizačním mechanismem. Pro authentizaci je nutno zadat údaje, které lze nastavit v souboru `conf/weblogin.conf`. Řešení používá HTTP Basic authentizaci. Jedná se o methodu authentizace relativně málo bezpečnou, poněvadž authentizační údaje jsou přenášeny s každým HTTP požadavkem ve formě kódování base64[11]. Pokud tedy není k dispozici alespoň zabezpečené spojení formou SSL/TLS, může se uživatel lehce stát obětí útoku Man-In-The-Middle. Zmíněná methoda pro authentizaci je pro účely této kvalifikační práce dostačující, avšak pro reálné nasazení řešení do provozu je vhodné tuto methodu zaměnit za některou z method více bezpečných, ať už ve formě moderní authentiazce stateless, či stateful.

5.3.2. Virtualizované rozhraní CLI

Funkcionalita, kterou nabízí pro ovládání démona klient CLI, je dostupná taktéž v identické podobě skrz rozhraní webové. Ve webovém rozhraní lze nalézt záložku s názvem Shell, pod kterou lze přistoupit k virtualizovanému rozhraní CLI ve webovém prohlížeči. Toto rozhraní je založeno na dynamických požadavcích XMLHttpRequest, které jsou zasílány v rámci SOP na `/proc` s GET parametrem `cmd` s hodnotou příkazu, který se má démonovi zaslat ke zpracování. Démonem je vrácena odpověď a ta je zobrazena ve virtualizované příkazové řádce. Webový shell je

[10] Označení server je zde použito v kontextu HTTP serveru, kde předmětný skript působí jako HTTP server, aby poskytoval součinnost pro připojení klientům HTTP skrz webový prohlížeč. V kontextu komplexnosti řešení (GGH) se však jedná o klienta, a takto bude i nadále v kapitolách kromě této označován, jelikož jako celek slouží k relativně krátkodobému poskytnutí funkcionality ovladatelnosti pro démona (server), který běží bez přerušení.

[11] Base64 není šifrování – je to druh kódování. Je to způsob reprezentace binárních dat v dobře zobrazitelné podobě.

převzat od Norberta Landsteinera – termlib.js.[17]

5.3.3. Teorie konfiguračního souboru a jeho editace

Struktura konfiguračního souboru je logická, postupuje se v definicích od obecného k určitému. Po učinění jakékoli změny v konfiguraci je potřeba démona restartovat, aby byl konfigurační soubor znovu načten. Načtení se tedy neprovádí automaticky.

Atributy platné všeobecně pro celý program jsou definovány na samotném začátku v části `[general]`. Zde jsou také definovány některé atributy týkající se chování pravidel pro jednotlivé služby, které jsou v konfiguračním souboru specifikovány níže. Platí zde mutatis mutandis pravidlo *lex specialis derogat legi generali*. To znamená, že pokud ten který atribut není definován níže speciálně pro konkrétní službu či pravidlo, použije se hodnota nastavená pravidlem obecným, tedy v části `[general]`. Nastavení v části `[general]` se považují za *leges generales*. Existují atributy, které nelze definovat globálně v rámci části `[general]`, stejně jako existují atributy, které lze definovat toliko právě tam.

Pravidla se navzájem nevylučují, ani neomezují. Fakt, že jeden log záznam, tedy jedna událost v journal souboru, vyhovuje definici pravidla jednoho, nevylučuje aplikaci pravidel ostatních, definovaných v konfiguračním souboru později. Lze zde hovořit o non-exkluzivitě pravidel. Dokonce ani fakt, že by jedno z pravidel mělo charakter pravidla všeobecného vůči jinému, které by v dáné situaci bylo relativně speciální (opět jde o vztah leges generales a leges speciales), nezpůsobuje neaplikaci kteréhokoli z nich. Pravidlo *lex specialis derogat legi generali* se tedy uplatňuje pouze pro definice atributů v části `[general]`, nikoli pro pravidla vůči sobě navzájem. Ty stojí na stejné horizontální rovině a jsou stejného významu.

Atribut `SOCKET_PATH` definuje cestu, kde bude vytvořen lokální linux socket pro komunikaci s klienty. Cesta může být udána absolutně nebo relativně. V případě relativní cesty je nutno považovat za pracovní adresář lokaci skriptu démona (`ggh_daemon.py`). Možnost užít způsob relativního definování cesty platí také pro všechny další atributy. Dále lze definovat atributy `HOSTS_DENY` a `HOSTS_ALLOW`, pomocí kterých lze apriorně vyloučit aplikaci pravidel pro určité IP adresy či dokonce rozsahy IP adres v podobě IP sítí. Jako hodnotu tyto atributy očekávají síť, tedy adresu sítě s lomítkem následovaným maskou sítě v notaci CIDR. Více sítí lze udat oddělením jednotlivých sítí čárkou. Dalším významným atributem je `DAEMON_SLEEP`, kterým lze definovat časové rozpětí mezi kontrolou nových logů v klasických journal souborech (ne journald). Pomocí `DB_EVENT_CHECK_SLEEP` lze určit, jak často se má provádět kontrola databáze, konkrétně tabulky `events`, za účelem „osvobození“ elementů (např. IP adres), které mají svůj trest již „odpykán“. Podrobněji je kontrola databáze rozebrána v kapitole 6.2.2.

5.3.4. Možnosti trasování chyb

Předností webového rozhraní je taktéž možnost trasování chyb, které nastaly za běhu démona. Takovéto chyby mohou být způsobeny chybou v samotném programu v oblasti, která nebyla dostatečně otestovaná, ale i chybným nastavením

konfiguračního souboru. To vše je dostupné pod záložkou s názvem Log ve webovém rozhraní. Pro přístupnost funkcionality trasování chyb z webového prostředí se předpokládá přístupnost journal souboru, který slouží k trasování chyb, z pozice webového klienta. Ve výpisu je možno sledovat nejen chyby, ale celkový průběh kontroly journal souborů a aktivitu démona. Měnit konfiguraci démona a pravidel detekce lze i skrze webové rozhraní. Algorithmus pro zabránění uložení konfiguračního souboru s nesprávným nastavením ve webovém rozhraní je ve stavu odevzdání této práce pouze ilustrativní povahy. Pouze ukazuje, jakým způsobem lze případné excesy v konfiguraci detekovat, neobsahuje však veškeré vhodné definice pravidel pro detekci chyby, které mohou teoreticky nastat.

Pokud je démon spuštěn v debug módu, trasování chyb v takovém případě není z webového prostředí dostupné. Průběh a případné chyby se vypisují přímo do příkazové řádky, tedy do `stdout`.

6. Technické aspekty implementovaného řešení

V této kapitole je obsažen popis vybraných aspektů, které jsou zajímavé kvůli své technické podstaty či implementace. Níže představené záležitosti by měly usnadnit orientaci ve zdrojovém kódu.

6.1. Linux démon

Démon je v běžné řeči označení pro počítačový program, který běží na pozadí operačního systému bez přímé interakce s uživatelem. Označení démon se užívá především v kontextu s operačními systémy založenými na GNU/Linux. Pro označení démona v prostředí Windows se častěji užívá názvu „služba“. Opatření serverové části software funkcionalitou a principy démona představuje v tomto případě fundamentální element adjustace pro praktické nasazení do provozu. Nasazení do provozu *à la longue* v klasickém módu, tedy tak, že program by byl navázán na TTY (terminál), je mimo testovací účely liché a nepraktické. Pro testovací účely obsahuje program přepínač `-d`, který zamezí přepnutí běhu programu do módu démona. K přepnutí běhu programu do módu démona je použita tzv. *double fork*[6] methoda.

Jako démon je schopen běžet nejen server, ale i webový server (`ggh_control_web.py`), který zprostředkovává webovou službu jakožto prostředek pro ovládání a kontrolu serveru skrz protokol HTTP.

Je nutno připomenout, že démon a případný klient, ať webový či CLI, musí být spuštěny se stejnými právy, aby bylo možno správně uskutečnit komunikaci mezi procesy pomocí unixového soketu.

6.1.1. Automatické spuštění při startu systému

V případě potřeby automatického spouštění démona současně se startem operačního systému je k dispozici v prostředí OS Linux hned několik možností, jak toho dosáhnout. První, pravděpodobně nejkorektnější a v dnešní době preferovanou metodou, je využití démona `systemd`, který má v posledních letech tendenci pomalu ale jistě nahrazovat systém `init.d`, který představuje druhou možnost zajištění automatického startu aplikace. Rozhodující je, který z těchto démonů pro správu systému je aktivní v hostitelském OS.

Pro automatizaci spuštění pomocí `systemd` je potřeba vytvořit tzv. service soubor (např. `ggh.service`) zpravidla v adresáři `/etc/systemd/system` s náležitým obsahem. Následně postačuje povolit službu v `systemd` např. pomocí příkazu `systemctl enable ggh`.

Další možností je kupříkladu démon `cron`, který v operačních systémech založených na systému Unix slouží k načasování spuštění pravidelných (zpravidla častěji se opakujících) úloh v konkrétním čase. Konkrétní čas zde může být nahrazen direktivou `@reboot`, která udává, že patřičný příkaz se má spustit ihned po startu systému.

6.1.2. Zabránění vícenásobným instancím

Je patrné, že nechceme připustit, aby uživatel / administrátor, třebaže i z nedbalosti, spustil program dvakrát, *nota bene* vícekrát. Pro zabránění vytvoření vícenásobných instancí stejného programu existují vícerá řešení. Jedná se o potřebu zajistit primitivní mechanismus IPC, tedy mechanismus meziprocesní komunikace, kde jeden proces dává vědet druhému, že běží.

Takový problém lze řešit mutexem, semaforem či zámkem. Jiná řešení nejsou vyloučena. Podrobný výklad těchto institutů by překračoval účel a rozsah této práce, a proto bude daná problematika vyložena pouze stručně. Semafory se užívají pro synchronizaci přístupu ke sdílené datové struktuře, přičemž dovolují více než jednomu vláknu (příp. procesu) z množiny vláken (procesů) na danou datovou strukturu v čase přistoupit. V tomto případě je problém interprocesní, nikoli intervláknový. Pojem mutex vykládají rozličtí autoři různě. Dle některých je mutex binární semafor, to znamená semafor s maximální hodnotou 1.[19][21] Jiní autoři na druhou stranu poukazují na rozdíly mezi mutexem a binárními semafory. Rozlišujícím prvkem přitom má býti to, že mutex je vlastněn procesem, který jej vytvořil, zatímco se semaforem se nepojí žádný vlastník.[28] Absence vlastnictví semaforu je takto údajně jeho pojmovým znakem. Hodnotu semaforu takto může měnit i proces, který je rozdílný od toho, který semafor původně uzamkl.[28] Shoda nicméně panuje přinejmenším na tom, že mutex disponuje jednou proměnnou, která se může nacházet ve dvou stavech – zamknuto a odemknuto.

Pro daný účel je vhodné využití zámku či mutexu. Mutex je sylabická zkratka anglického sousloví „mutual exclusion“, do češtiny překládáno jako vzájemné vyloučení. Institut mutexu i zámku vznikl proto, aby současně běžící procesy popř. vlákna nemohly nekoordinovaně simultánně pozměňovat jednu a tutéž datovou strukturu, což by mohlo mít za následek, že taková datová struktura by se po-

tenciálně mohla dostat do nekonzistentního stavu. Použití institutu mutexu pro zajištění současného běhu právě jedné instance programu je podružný způsob jeho využití. Zde totiž nejde o udržení konzistence, ale o tzv. globální (systemwide) mutex, který v našem případě slouží toliko jako vztažný bod pro kontrolu existence instance programu. Linux není nakolněn tzv. systemwide mutexům, jaké jsou dobře známé např. z OS Windows a jeho WinAPI (`CreateMutex()`). Python knihovna mutex je od verze Python 2.6 označena za obsoletní a pro Python 3.x byla dokonce zcela odstraněna.[30] Navíc se jedná o funkcionalitu intervláknové synchronizace, nikoli interprocesní[12], a proto nebude využita. Zámek – funkce `Lock` podle definice v knihovně `multiprocessing` – je pro zde potřebné využití nevhodná.

Jelikož program pro účely konvenienční terminace démona (pomocí příkazu `ggh_daemon.py stop`) zapisuje ID svého procesu (PID) do souboru, využijeme tento soubor jako vztažný bod pro určení existence běžící instance programu. Vhod nám zde příjde knihovna `fcntl`, která představuje wrapper kolem Linuxu inherentního systémového volání `fcntl()`. `Fcntl()` je zkratkou anglického file control. Umožnuje nám provádět operace nad otevřenými deskriptory souborů.[5] Budeme tedy zamykat soubor, který když se pokusí zamknout případná druhá instance programu, pokus o takové zamknutí selže a program terminuje s návratovou hodnotou False.

Výpis № 5 Extrakt kódu z implementace demonstrující logiku využití souboru jako vztažného bodu pro určení existence již běžící instance programu

```
1  def lockFile(self, pidfile, timeout=0.5):
2     try:
3        fd = os.open(pidfile, os.O_CREAT)
4     except Exception:
5        print("Opening pidfile failed.")
6     flags = fcntl.fcntl(fd, fcntl.F_GETFD, 0)
7     flags |= fcntl.FD_CLOEXEC
8     fcntl.fcntl(fd, fcntl.F_SETFD, flags)
9     started = time.time()

11    while True:
12       try:
13          fcntl.flock(fd, fcntl.LOCK_EX | fcntl.LOCK_NB)
14          return True
15       except Exception:
16          if started > time.time() - timeout:
17             print("Couldn't obtain lock but we're within timeout.")
18             pass
19          else:
20             print("Waiting time exceeded.")
21             return False
22             break # unnecessary but cool
23       time.sleep(0.1)
```

[12] V terminologii mutexů a příbuzných mechanismů panuje poněkud velký chaos, lze jen ztěží činit generalizující závěry.

```
25 def doNotRunTwice(self):
26    if self.lockFile(self.pidfile) == False:
27       pf = open(self.pidfile, 'r')
28       pid = int(pf.read().strip())
29       if pid:
30          sys.stderr.write("GGH is already running under PID [%s]" %
               ↪ pid)
31       pf.close()
32       sys.exit(1)
```

6.2. Databáze

Databáze je postavena na python modulu `sqlite3`. Veškeré funkce ohledně práce s databází jsou definovány ve zdrojovém souboru `database.py`. Databáze obsahuje dvě tabulky – `events` a `eventlog`.

6.2.1. Tabulka events a eventlog

Databáze, jak již bylo zmíněno výše, obsahuje dvě tabulky. Do tabulky `events` se ukládají události, které naplnily podstatu definice patřičného pravidla a takové pravidlo obsahuje nikoli pouze akci, ale i antiakci (atribut `ANTIACTION`), která se má vykonat po skončení doby definované v atributu `JAILTIME`. Pokud je řeč o vykonání `ANTIACTION`, lze v tomto významu hovořit o tzv. reintegraci[34] adresy IP. Do tabulky `eventlog` se na druhou stranu ukládají metadata o všech detekovaných událostech na základě pravidel, a to i těch, které fakticky nevyvolaly žádné následky (nebyl u nich nastaven atribut `ACTION`) nebo které nemají po skončení doby „trestu“ (`JAILTIME`) provést žádná (často nápravná) opatření – atribut `ANTIACTION`).

Důležitou roli zde hraje atribut `DO_NOT_DUPLICATE`, který může být nastaven v části `[general]` nebo u jednotlivých pravidel samostatně. Pokud je nastaven na hodnotu true, postará se démon o to, aby nevykonával (neprováděl příkaz v `ACTION`) pro elementy, které již byly detekovány dříve a doposud jim neskončila doba trestu (`JAILTIME`). Pro posouzení identity elementu se zde bere v potaz dvojice `RULENAME` a hodnota, kterou nabývá tzv. distingueur. Distingueur je řetězec, který se získá extrahováním patřičného řetězce pomocí tzv. Named Group definovaného v atributu `CRITERIA_TO_DISTINGUISH`, je-li to relevantní.

V duchu rozlišování povahy metadat o útocích v tabulkách `events` a `eventlog` se nese i pojmosloví v případě nakládání s daty v těchto tabulkách. Pro vymazání záznamu z tabulky `eventlog` se v klientu CLI zadává příkaz `db eventlog remove`, zatímco v případě tabulky `events` to je `db events release`. Slovo release v sobě totiž nese nejenom příkaz k odebrání záznamu z databáze, ale i učinění potřebných kroků k zahlazení následků způsobených vykonáním příkazu v `ACTION` – tedy provedení příkazu v `ANTIACTION`.

Pro úplnost lze ještě zmínit, že pokud je výše zmíněný distingueur adresa IP, lze se pokusit skrz webového klienta tuto IP adresu převést zpětným DNS lookupem

Vyobrazení № 1 Zpětný DNS lookup ve webovém prostředí klienta

8	test	test_rule1	⊞ 185.234.219.51	Sat Mar 30 01:44:50 2019	86400
9	test	test_rule1	⊞ 189.84.211.117 189.84.211-117.dinamicatelecom.net.br	Sat Mar 30 01:44:50 2019	86400
10	test	test_rule1	⊞ 185.234.216.235	Sat Mar 30 01:44:50 2019	86400

na doménové jméno. Pokud zpětný DNS lookup uspěje, lze takto velice snadno alespoň odhadnout, z kterého státu IP adresa je. Podstatným přitom nemusí být toliko doména prvního řádu (TLD). Zpětný DNS lookup může vypadat například jako ve vyobrazení č. 1.

6.2.2. Pravidelná kontrola stavu databáze

Jak již bylo zmíněno výše, pomocí atributu `DB_EVENT_CHECK_SLEEP` v konfiguračním souboru lze určit, jak často se má provádět kontrola tabulky `events` za účelem „osvobození“ IP adres, s již „odpykaným“ trestem. V souboru `rule_executor.py` je definovaná třída `DbWatcher`, která rozšiřuje třídu `Thread`. Třída `DbWatcher` se vyjímá svou konstrukcí. Jedná se o vlákno, které je při spuštění démona načasováno třídou `scheduler` ke spuštění za dobu udanou v `DAEMON_SLEEP`. Jakmile je vlákno pro kontrolu databáze spuštěno, načasuje samo sobě své další spuštění za daný časový úsek a teprve potom provede kontrolu databáze. Další zajímavostí je, že kromě toho lze databázi zkontrolovat kdykoli ručně, jak již bylo zmíněno v kapitole 5.2. O toto se stará třída `Queue`.

6.3. Konfigurační soubor a jeho zpracování

Konfigurační soubor a jeho struktura, včetně struktury definic služeb a jejich pravidel je ražen maximou maximální versatility, jakožto základní myšlenkou této práce. I přesto však není strukturou v užším slova smyslu (z hlediska gramatiky) pojednávaný soubor nic ojedinělého či nevídaného. S velmi podobnou strukturou pracuje dokonce třída `ConfigParser` ze stejnojmenného modulu. Důvodem pro nevyužití této třídy pro zpracování nastavení v konfiguračním souboru byla potřeba mít plnou kontrolu nad jednotlivými částmi konfigurace. Základní zpracování probíhá podle tříd a method ve zdrojovém souboru `config.py`. Základní je zde třída `Prefs`, která se stará o čtení konfiguračního souboru, převádění jednotek času v atributech na sekundy, a v neposlední řadě naplňuje objekty tříd `Rule`, `Service` a `ServicesManager` náležitými daty. Třída `ServicesManager` se stará o uchování struktury jednotlivých služeb v paměti. Struktura objektů služeb je potom dána třídou `Service`, která zařizuje uchování informací náležících službě jako celku, nikoli jednotlivým pravidlům, a udržuje odkazy na jednotlivá pravidla k dané službě náležející. Struktura jednotlivých pravidel je nakonec specifikována třídou `Rule`.

Třída `Prefs` je postavena na návrhovém vzoru jedináček (singleton). Toto umožňuje, že na instancované objekty uvnitř objektu třídy `Prefs` lze přistupovat z re-

lativně nezávislých míst v kódu a opětovným voláním Prefs() nedochází k opětovnému čtení konfiguračního souboru, nýbrž je vrácena reference na původní instanci třídy. Vždy tedy může současně existovat pouze jedna instance třídy. Realizace návrhového vzoru je ve výpise č. 6.

Výpis № 6 Extrakt kódu ze třídy Prefs zajišťující funkcionalitu návrhového vzoru jedináček

```
1     class Prefs(): RULENAME
2        # Keep instance reference
3        _singletonInstance = None
4        _configFile = None
5
6        def __new__(cls, *args, **kwargs):
7           if not cls._singletonInstance:
8              # Create instance
9              cls._singletonInstance = super(Prefs, cls).__new__(Prefs)
                   ↪  # object
10             pass
11          # Return the instance
12          return cls._singletonInstance
13
14       def __init__(self, configFile=None):
15          if configFile == None:
16             return
17
18          self.load_prefs(configFile)
19          pass
20
21       @staticmethod
22       def getInstance():
23          return Prefs._singletonInstance
```

6.4. Komunikační protokol na bázi unixového socketu

Komunikační protokol, který poskytuje spojení mezi démonem a klienty pro ovládání démona, je realizován pomocí lokálního unixového socketu (`AF_UNIX`). Inspirací pro využití právě této techniky byl projekt Fail2Ban. Část kódu z něj je použito v rámci zákonné citační výjimky (§ 31 odst. 1 písm. a) zákona č. 121/2000 Sb., autorský zákon) přímo v řešení. Na straně serveru (démona) je použito třídy `dispatcher` z modulu `asyncore`. Komunikačnímu mechanismu je věnováno celé samostatné vlákno. Na straně klientů (klient webový i CLI) je použito základního modulu `socket` a method `recv` pro přijímání zpráv a `send` pro odesílání.

6.5. Princip kontroly klasických journal souborů

Klasickými soubory journal se rozumí textové soubory, kam se zapisují události (transakce), přičemž jedné službě je zpravidla věnován jeden soubor. Agregace vícero služeb do souboru jednoho je ojedinělá. Tyto soubory jsou GGH čteny buď retroaktivně – v případě atributu `RETROACTIVE` nastaveného na `true`, nebo jinak ve výchozím nastavení je po spuštění démona pomoci funkce `seek` souborový ukazatel načten na konec souboru journal a v pravidelných intervalech (určeným atributem `DAEMON_SLEEP`) je kontrolováno, zdali se do souboru nepřipsal nový log (nová událost). Jakmile je nalezen nový log, provede se na něj kontrola všech pravidel.

6.6. Princip kontroly souboru journald

Kontrola systému journald probíhá podstatně jinak. Pro zpracování zpráv journálu systemd je použit python modul `systemd`. Pomocí methody `poll` je démon ihned informován, pokud se do journald zapíše nová zpráva. Nevýhodou tohoto přístupu je, že se pro každou službu (nikoli pravidlo) vytváří separátní vlákno, které si takto hlídá svoji službu. Separátního vlákna je potřeba proto, aby methoda `poll` neblokovala hlavní vlákno. Rovněž lze využít atributu `RETROACTIVE` nastaveného na true. V takovém případě se pravidla aplikují i na události zaznamenané v journald ještě před spuštěním démona.

7. Pravidla detekce vybraných služeb

Cílem práce bylo mimo jiné zamyslet se nad možnými útoky na v zadání vymezené služby a navrhnout pro ně detekční pravidla. Při uvažování o možných útocích byl brán v potaz fakt, že řešení slouží primárně jako všeobecný prostředek na ochranu blíže nespecifikovaného okruhu služeb, přičemž pro mnoho konkrétních služeb existují řešení specializovaná, která jsou lépe adaptovaná pro konkrétní službu a jsou tak schopna poskytnout granulárnější methody detekce případných útoků. Jako příklad zde lze uvést například přídavný modul `mod_security` pro webový server Apache, který zabezpečuje, pokud je nakonfigurován správně, velice dobrou ochranu pro celou řadu nejrůznějších útoků. Je schopen detekovat útoky, které z běžných logů serveru Apache nejsou detekovatelné. Apache totiž například nezaznamenává do journal souborů informace přenášené pomoci methody `POST`, ve které se může skrývat hned několik potenciálních útoků. V případě modulu `mod_security` si však lze velice dobře představit vzájemnou spolupráci s předkládaným řešením, k tomu blíže kapitola 7.2. Velmi zde záleží na konkrétním scénáři, některé útoky lze i navzdory tomuto faktu např. na základě odpovědi serveru i přesto detekovat. Při vytváření pravidel v předkládaném řešení je potřeba dbát methodologie, že předkládané řešení slouží především jako subsidiární prostředek ochrany, a to především služeb, které nemají k dispozici specializované možnosti pro zabránění útokům. Prakticky, avšak methodologicky ne zcela správně, přichází v úvahu také využití předkládaného řešení za účelem možnosti definovat pravidla pro veškeré služby centralizovaně a spravovat je takto pohodlně na jednom místě.

7.1. SSH

Služba SSH umožňuje zabezpečenou komunikaci mezi dvěma body v síti prostřednictvím zprostředkování vzdáleného přístupu k příkazové řádce. SSH nabízí ve srovnání se staršími a méně bezpečnými technologiemi mnoho způsobů authentizace. Kromě klasické authentizace heslem je zde možnost využití veřejného klíče, dále tzv. host-based authentizaci, interaktivní keyboard authentizaci a další způsoby.[38]

V tomto smyslu lze také rozličně definovat pravidla detekce. Lze rozumně předpokládat, že pokud se potenciální útočník rozhodne pro útok na protokol SSH, bude se zpravidla jednat o útok hrubou silou a bude zaměřen na authentizační methodu heslem (methoda `password`). Jen stěží si lze představit útok hrubou silou na methodu authentizace veřejným klíčem. Pravidlo detekce omezené toliko na methodu password je uvedeno ve výpisu č. 7.

Výpis № 7 Pravidlo pro detekci neúspěšné authentizace methodou password ke službě SSH

```
# This rule is restricted to detect solely the "password" ssh method
4_RULENAME = x_ssh_failed_login1
4_ENABLED = true
4_CRITERIA_TO_DISTINGUISH = adresseIP_ggf
4_REGEX = "^Failed password for (?P<user>.*) from (?P<adresseIP_ggf
    ↪ >[^ ]*) port"
4_THRESHOLDCOUNT = 5
4_ACTION = "iptables -I INPUT -s adresseIP_ggf -p tcp --destination-
    ↪ port 22 -j DROP"
4_JAILTIME = 1d
4_ANTIACTION = "iptables -D INPUT -s adresseIP_ggf -p tcp --
    ↪ destination-port 22 -j DROP"
```

Pokud by bylo z nějakého důvodu žádoucí detekovat všechny methody jako relevantní pro incident, pravidlo by mohlo vypadat, jak je uvedeno ve výpisu č. 8.

Výpis № 8 Pravidlo pro detekci neúspěšné authentizace kteroukoli methodou

```
# Detects failed authentication attempts regardless what auth.
    ↪ method has been tried
3_RULENAME = x_ssh_failed_login2
3_ENABLED = true
3_CRITERIA_TO_DISTINGUISH = adresseIP_ggf
3_REGEX = "Failed (?P<method>\S*) for (?P<invalid>invalid user |
    ↪ illegal user )?(?P<user>.*) from (::ffff:)?(?P<adresseIP_ggf
    ↪ >[^ ]*)( port \d+)?( ssh2)?$"
3_THRESHOLDCOUNT = 5
3_ACTION = "iptables -I INPUT -s adresseIP_ggf -p tcp --destination-
    ↪ port 22 -j DROP"
3_JAILTIME = 1d
3_ANTIACTION = "iptables -D INPUT -s adresseIP_ggf -p tcp --
    ↪ destination-port 22 -j DROP"
```

7.2. Apache

Pokud jde o webový server Apache, je vhodné zmínit, že na webové aplikace existuje nepřeberné množství nejrůznějších útoků. S tím přichází celá řada možných pravidel detekce. V duchu správné methodologie, jak je nastíněno výše, se pro účel této práce uvedou pravidla toliko dvě.

Prvním z nich je pravidlo, které zajistí omezený počet pokusů HTTP Basic authentizace. Pravidlo je založeno na tom, že při neúspěšné authentizaci je odpověď serveru Apache formou HTTP status kódu 401. Nejde tedy o nic jiného, než že se detekují status kódy 401 a jakmile je překročen jejich hraniční počet definovaný v atributu `THRESHOLD` pro konkrétní adresu IP, je proveden příkaz v atributu `ACTION`. Pravidlo se v níže uvedeném příkladě vztahuje na všechny HTTP methody, lze však v případě potřeby tuto část pravidla libovolně upravit dle pořeby `ad hoc`.

Výpis № 9 Pravidlo pro detekci neúspěšné authentizace kteroukoli methodou

```
200_RULENAME = apache_http_auth
200_ENABLED = true
200_CRITERIA_TO_DISTINGUISH = adresseIP_ggf
200_REGEX = "^(?P<adresseIP_ggf>[^\s]+) .* "(GET|POST|OPTIONS|PUT|
    ↪ DELETE|TRACE|PATCH|HEAD|CONNECT) (?P<url>.*) HTTP/[012].[012]"
    ↪  401"
200_THRESHOLDCOUNT = 5
200_ACTION = "iptables -I INPUT -s adresseIP_ggf -p tcp --
    ↪ destination-port 80 -j DROP"
200_ANTIACTION = "iptables -D INPUT -s adresseIP_ggf -p tcp --
    ↪ destination-port 80 -j DROP"
```

Zobecněním výše uvedeného pravidla lze uvažovat nad pravidlem poněkud rigorózněji. Lze totiž rozšířit působnost detekce na všechny status kódy kategorií 4xx (chyba na straně klienta) a 5xx (chyba na straně serveru). Typicky je navrácen status kód 404 v případě neexistujícího dožádaného zdroje. Může se často jednat například o případy použití automatizovaných nástrojů pro hledání přihlašovací stránky administrátora pro následné užití bruteforce útoku či využití jinak získaných authentizačních údajů k zajištění dlouhodobého přístupu do aplikace např. nahráním tzv. shellu na server. Dále je relativně velké množství útoků, které způsobují vrácení chybového status kódu 5xx. Vyloučena není ani součinnost vícero aplikací na obranu proti útokům. Status kód 5xx vrací mimo jiné přídavný modul pro Apache `mod_security`, který je schopen webovou aplikaci bránit např. proti pokusům o útoky LFI, RFI, SQLi a spoustu dalších. Velice běžným scénářem je, že dříve či později se i navzdory nasazenému modulu `mod_security` útočníkovi podaří najít správně „obfuskovaný“ požadavek, který obejde pravidla `mod_security` a útočníkovi se tak nakonec podaří dosáhnout svého cíle. Nasazením předkládaného řešení lze útočníka zastavit nebo minimálně zkusit odradit ještě dříve, než se mu podaří ochranu `mod_security` obejít. Příklad pravidla je uveden ve výpisu č. 10.

Výpis № 10 Pravidlo pro detekci neúspěšné authentizace kteroukoli methodou

```
100_RULENAME = apache_statuscode
100_ENABLED = true
```

```
100_CRITERIA_TO_DISTINGUISH = adresseIP_ggf
100_REGEX = "^(?P<adresseIP_ggf>[^\s]+) .* "(GET|POST|OPTIONS|PUT|
    ↪ DELETE|TRACE|PATCH|HEAD|CONNECT) (?P<url>.*) HTTP/[012].[012]"
    ↪  (?P<statuscode>[4-5]\d\d)"
100_THRESHOLDCOUNT = 50
100_ACTION = "iptables -I INPUT -s adresseIP_ggf -p tcp --
    ↪ destination-port 80 -j DROP"
100_ANTIACTION = "iptables -D INPUT -s adresseIP_ggf -p tcp --
    ↪ destination-port 80 -j DROP"
100_VAL = 1d
```

7.3. Postfix a Dovecot

Postfix a Dovecot jsou software zajišťující služby SMTP resp. IMAP a POP3 pro fungování elektronické pošty. Pro účely této práce byly vypracovány pravidla pro detekci opakované neúspěšné authentizace k oběma službám. Pro detekci sofistikovanějších útoků existuje celá řada specializovaných nástrojů, které svou roli hrají významně lépe než případné pravidla vytvořené v rámci řešení GGH. Zpravidla se v případě služeb Postfix a Dovecot jedná o specializovaný software pro odhalování spamu. Opakovaně neúspěšnou authentizaci ke službě Postfix dokáže odhalit pravidlo uvedené ve výpisu č. 11.

Výpis № 11 Pravidlo pro detekci neúspěšné authentizace k SMTP serveru Postfix

```
1_RULENAME = postfix_auth
1_ENABLED = true
1_CRITERIA_TO_DISTINGUISH = adresseIP_ggf
1_REGEX = ".*: warning: unknown\[(?P<adresseIP_ggf>.*?)\]: SASL
    ↪ LOGIN authentication failed: authentication failure"
1_THRESHOLDCOUNT = 5
1_ACTION = "iptables -I INPUT -s adresseIP_ggf -p tcp -j DROP"
1_JAILTIME = 1d
1_ANTIACTION = "iptables -D INPUT -s adresseIP_ggf -p tcp -j DROP"
```

Pravidlo pro detekci opakované neúspěšné authentizace k Dovecot zde není uvedeno samostatně, protože je velmi podobné pravidlu pro Postfix. Lze ho však nalézt v příloze D.

8. Právní aspekty zpracovávání osobních údajů

Ochrana osobních údajů je v dnešní době integrální součástí společensko-technologického vývoje a její zásady by měly být brány v potaz během každého vývoje software.

S ohledem na expansivní místní a potažmo osobní působnost nařízení GDPR (dále jen Nařízení) lze říci, že požadavky na ochranu osobních údajů stanoveny

tímto nařízením by neměly být zanedbány. Je racionální pracovat zde pouze s nařízením GDPR, jelikož lze rozumně předpokládat, že tento předpis bude v této oblasti relevantní pro většinu administrátorů, kteří tuto práci/řešení případně upotřebí či rovnou nasadí do provozu. Dále lze s jistou odvahou říci, že standardy nastolené Nařízením jsou natolik vysoké, že dostáním povinnostem vyplývajícím z tohoto nařízení lze očekávat, že bude učiněno zadost i většině jiným právním řádům mimo působnost nařízení v kontextu ochrany osobních údajů.

Věcná a místní působnost Nařízení je upravena v článcích 2 a 3. Nařízení stanovuje, že za podmínky, že se zpracovávají osobní údaje subjektů údajů, které se nacházejí na území EU, se toto Nařízení použije i pro zpracování osobních údajů, ke kterému dochází mimo území EU.[27] Podle čl. 2 odst. 2 nedopadá Nařízení na zpracování, ke kterému dochází při výkonu činností, jež nespadají do oblasti působnosti práva EU, ani na zpracování prováděné za účelem prevence, vyšetřování, odhalování či stíhání trestných činů. Z působnosti Nařízení je dále vyňato zpracování osobních údajů, které provádí fyzická osoba výlučně pro své osobní nebo domácí činnosti.[27] K aplikaci Nařízení tedy dochází v situaci, kdy ke zpracování osobních údajů dochází v souvislosti s činností provozovny správce nebo zpracovatele, která se nachází na území EU a dále v případech, kdy zpracování probíhá mimo EU, ale zpracovávají se osobní údaje subjektů údajů, které se nacházejí v EU.[27] Teoreticky k aplikaci Nařízení dojde také tehdy, pokud se českého právního řádu (nebo právního řádu jiného členského státu EU) má užít na základě mezinárodního práva soukromého. Správci a zpracovatelé (kdo to v kontextu této práce je, je uvedeno níže) si musí vyhodnotit, zdali se na ně Nařízení vztahuje. Pokud ano, je potřeba aby průběh a podmínky zpracování adaptovali požadavkům, které toto Nařízení stanovuje.[27]

Osobními údaji, jak udává čl. 4 Nařízení, jsou veškeré informace o identifikované nebo identifikovatelné fyzické osobě.[24] Tím mohou být například jméno, identifikační číslo, lokační údaje, síťový identifikátor (např. IP adresa), informace o zálibách, vlastnostech, názorech, majetkových poměrech, vztahu osoby k jiným lidem nebo jiné zvláštní prvky fyzické, fysiologické, genetické, psychické, ekonomické, kulturní nebo společenské identity fyzické osoby. Je tedy patrné, že ne všechna zpracovávaná data jsou osobními údaji.

Osobním údajem, který je v kontextu této práce nejvíce relevantním, je bezesporu adresa IP. Vnímání IP adresy jakožto osobního údaje je už delší dobu předmětem neshod. Zatímco o statické IP adrese jako osobního údaje nikdy větší spory nebyly, otázku povahy IP adresy dynamické směrodatně osvětlil teprve v roce 2016 SDEU v případě Breyer[32]. V případě dynamické adresy jde totiž o to, že se téměř po každém opětovném připojení uživatele do sítě mění. Je tedy otázkou, zdali taková adresa může vést k identifikaci osoby. Existují dva pohledy, mluvíme o tzv. absolutním a relativním měřítku.[12] Teorie absolutního měřítka bere v potaz, že identifikace osoby se může uskutečnit buď provozovatelem webové služby nebo subjektu třetího – např. poskytovatelem internetového připojení – pokud jsou k tomu dány právní prostředky. Ve smyslu takového chápání identifikovatelnosti je osoba identifikovatelná přinejmenším nepřímě, a tudíž představuje i dynamická IP adresa osobní údaj. Na druhé straně u teorie měřítka relativního jde o to, zdali samotný provozovatel webové služby je schopen vytvořit konkrétní vazbu mezi IP adresou a uživatelem.[12] Německý spolkový soudní dvůr zastával pohled absolut-

ního měřítka.[31] Po předložení předběžné otázky se však Soudní dvůr evropské unie nechal slyšet, že ukládání IP adresy jakožto osobního údaje je v souladu s evropským právem pouze tehdy, pokud slouží k zajištění všeobecné funkcionality služeb – například k efektivní obraně proti útokům. Je však potřeba při zpracovávání takových osobních údajů vždy dbát na proporcionalitu s oprávněnými zájmy a základními právy a svobodami uživatele (subjektu údajů).[32]

Právním základem zpracování IP adresy a jiných údajů, které předkládané řešení (ale i všechny konkurenční produkty uvedené v kapitole 4) provádí, je čl. 6 odst. 1 písm. f) Nařízení, protože zpracování údajů je zde nezbytné pro účely oprávněných zájmů správce osobních údajů.

Dříve než se budou diskutovány konkrétní právní otázky v předkládaném řešení, je vhodné připomenout implementaci DenyHosts z kapitoly 4.2. DenyHosts jako producent software zde provozuje webovou stránku, na které jsou dostupné kromě jiného IP adresy potenciálních útočníků, které DenyHosts (všechny jeho instalace) označil za podezřelé. Tato webová stránka je volně dostupná všem uživatelům Internetu. V takovém případě je správcem provozovatel webové stránky, tedy DenyHosts, jelikož určuje účel a prostředky zpracování osobních údajů (čl. 4 odst. 7 Nařízení). Uživatel software je zpracovatelem. Je možné, že by uživatel software a provozovatel webové stránky mohli být kvalifikováni jako společní správci ve smyslu čl. 26 Nařízení. Toto by přicházelo především do úvahy v případě, pokud by zde existovala *opt-in* možnost aktivace funkcionality pro odesílání dat na onen centralizovaný web. To však není případ DenyHosts. DenyHosts nesprávně reflektuje jedny z fundamentálních principů ochrany osobních údajů, a to *privacy by design* a *privacy by default*, které jsou zakotveny v čl. 25 odst. 1 Nařízení. Poněkud jiná situace by byla, pokud by web zpracovával osobní údaje ve formě IP adres pro účely vizualizace, odkud útoky nejčastěji pocházejí a popř. kam směřují. V takovém případě by ale na druhou stranu bylo zbytečné konkrétní IP adresy na webu zveřejnovat. V předkládaném řešení je však určení, kdo je správcem a zpracovatelem, relativně očividné. Za běžných okolností to bude subjekt, jež řešení nasadil do provozu.

Zásada *privacy by design* znamená ochranu osobních údajů pomocí stavu techniky a technických řešení. Tato maxima sahá po základní myšlence, že ochrany osobních údajů lze nejlépe dostát, pokud je ochrana technicky integrována a zaručena již při procesu automatizovaného zpracovávání. Jinými slovy, ochrana osobních údajů ve smyslu zásad nařízení by měla vždy začínat včasným uchopením technických a organizačních opatření již ve stádiu vývoje. Toho je v případě předkládaného řešení dosaženo například nutnosti authentizace pro přístup k webovému rozhraní.

Maxima *privacy by default* je speciální částí zásady *privacy by design.*[9] Jedná se o upřednostnování ochrany osobních údajů pomocí výchozích přednastavení (software) před jinými hodnotami. Ve smyslu této základní myšlenky mají být chráněni především uživatelé, kteří jsou s daným software méně zdatní než producent sám, a nejsou tudíž s to přizpůsobit si nastavení v souladu s maximální ochranou osobních údajů. Tato myšlenka stojí za pojmem „privacy paradox“, podle kterého uživatelé o své soukromí dbají a upřednostnují ho, aktivně se však takto již nechovají.[13] Výrazem zásady *privacy by default* je v této práci například to, že

webové rozhraní se nespouští automaticky se spuštěním démona, nýbrž je potřeba ho spustit separátně.

Předkládané řešení nedisponuje žádnou centralizovanou webovou stránkou, kde by se agregovaly informace o hlášených potenciálních incidentech ze všech instalací. Řešení poskytuje funkcionalitu pro přehled detekovaných potenciálních útoků ve webovém prostředí primárně pro interní potřeby správy, tedy administrátora (uživatele) a dalším povolaným subjektům. Výchozí nastavení řešení, jak je popsáno výše, je pro dostání hodnot právní úpravy ochrany osobních údajů velice relevantní. Z tohoto důvodu disponuje každá instalace řešení, respektujíc maximu *privacy by default*, *opt-in* možností spuštění webového rozhraní (viz kapitolu 5.3). Toto webové rozhraní je standardně opatřeno authentizací, která chrání osobní údaje před přístupem nepovolaných subjektů. Na administrátorovi je ponechána volba, zdali učiní potřebné kroky, jakými jsou např. nastavení náležitých parametrů firewallu či port forwardingu na směrovačích příp. stanicích, aby se webová služba stala dostupnou nejen v rámci sítě vnitřní, ale také vnější (Internetu).

8.1. Odpovědnost a právo na náhradu újmy

V souladu s čl. 82 odst. 1 Nařízení má kdokoli, kdo v důsledku porušení Nařízení utrpěl hmotnou či nehmotnou újmu, právo obdržet od správce nebo zpracovatele náhradu utrpěné újmy. To znamená, že jak správce, tak i zpracovatel jsou pro účely náhrady újmy pasivně legitimováni. Jak udává čl. 82 odst. 4 Nařízení, každý správce nebo zpracovatel nese odpovědnost za celou újmu, a to z důvodu, aby aby byla zajištěna účinná náhrada újmy subjektu údajů. Primární odpovědnost má vždy správce osobních údajů. Mezi správcem a zpracovatelem existuje právo regresu zakotvené v čl. 82 odst. 5 Nařízení. Nastane-li tedy situace, že některý správce či zpracovatel zaplatí plnou náhradu způsobené újmy, má právo žádat od ostatních správců či zpracovatelů vrácení části náhrady odpovídající jejich podílu na odpovědnosti.

Existují názory, že z občanskoprávní odpovědnosti za způsobenou újmu v souvislosti s ochranou osobních údajů se lze dle českého občanského zákoníku, byť jen částečně, liberovat, oznámí-li standardně odpovědný subjekt omezení své odpovědnosti ještě před vznikem takovéto újmy – věta druhá § 2896 zákona č. 89/2012 Sb., občanský zákoník. Toto je však sporné v souvislosti s aplikační předností práva EU před právem vnitrostátním. Správní odpovědnosti se však zbavit nelze.

9. De emendatione futura aneb výzvy budoucnosti

Ačkoli práce zde předkládaná představuje funkční implementaci řešení, je nepochybné, že teoretické možnosti jeho rozšíření či zefektivnění v budoucnu nejsou ani zdaleka vyčerpány. Tato kapitola popisuje rozvahy o možnostech budoucího vývoje řešení, které jsou nad rámec této práce.

9.1. Výzvy statické a dynamické detekci útoků

Užitečným zlepšením do budoucna by mohlo být inkrementální zvyšování doby blokace (`JAILTIME`) potenciálních útočníků (často IP adres, ze kterých je útočeno). Důvod je zřejmý – s narůstající četností výskytu události, které se dají podřadit pod pravidlo detekce, roste významně pravděpodobnost, že se jedná o skutečný útok a nikoli toliko o náhodné nedopatření nic netušícího uživatele služby. Vhodné by zde bylo dobu blokace IP adresy (či jiného trestu) zvyšovat se vzrůstajícím počtem detekovaných údálostí. Uvažovat zde lze o zvyšování lineárním až exponenciálním v ohledu na počet udělených trestů v minulosti.

Jak již bylo nastíněno v kapitole 3.1.2, účinnost ochrany by mohla být markantně zvýšena implementováním algoritmů strojového učení. Mohlo by tak být dosaženo flexibilní ochrany pro neznámé či neočekávané typy útoků.

9.2. Granulární nastavení pro rozlišení prvků v pravidlech

Každé pravidlo může v současné podobě řešení obsahovat jeden parametr `CRITERIA_TO_DISTINGUISH`, jehož hodnota určuje název tzv. `named group` v regulárním výrazu (atribut `REGEX`), který je poté směrodatný pro rozlišení kritéria, na jehož základě jsou jednotlivé logy (detekované útoky) řazeny k sobě pro účely sledování dovršení hodnoty `THRESHOLD` a následné aplikace akce pravidla (`ACTION`), které obsahuje systémový příkaz zajišťující vhodou reakci na detekovaný útok. Atribut `ACTION` často, nikoli však nezbytně nutně, obsahuje rovněž hodnotu `CRITERIA_TO_DISTINGUISH` – často se zde bude jednat o adresu IP, vůči které se má uplatnit opatření v systémovém firewallu.

V současné době takto postavené pravidlo neposkytuje maximální flexibilitu pro jemné nuance kritérií pro detekci útoku. V praktické rovině by se mohla jevit za jistých okolností užitečnou např. možnost počítání jednotlivého pravidla pro authentizaci k webovému serveru Apache (viz kapitolu 7.2) zvlášť v závislosti na tom, k jakému dožádanému zdroji se klient chce authentizovat. Neúspěšné pokusy o HTTP Basic authentizaci ke zdroji `/employees` by se nepočítaly v rámci stejného čítače výskytu jako neúspěšné pokusy o přihlášení ke zdroji `/admin`. Nemožnost docílit takové míry granulárnosti pravidel detekce je v rozporu se zásadou maximální versatility, a proto by bylo vhodné tento nedostatek v budoucnu odstranit.

9.3. Výpočet síly útoku a diversifikace opatření

Vyšší granularity nastavení lze dále dosáhnout, pokud by byl implementován algoritmus pro výpočet síly útoku a pravděpodobnosti, že se opravdu jedná o útok. V závislosti na tom by poté bylo možné udělit rozličné sankce. Sankce by mohly být rozděleny do několika stupňů ve formě separátních `ACTION` atributů (mohlo by se jednat např. o `ACTION_LOW`, `ACTION_MEDIUM` a `ACTION_CRIT`). Každá takto definovaná akce by odpovídala stupni závažnosti (síle útoku). Je vhodné zablokovat IP adresu, která se snaží authentizovat desetkrát během dvou vteřin a jinak tu,

která tak učiní během pěti minut.

9.4. Reakce pravidla na úspěšnou událost

Další užitečnou záležitostí, kterou lze v budoucnu implementovat, by mohlo být vynulování čítače logů odpovídajích pravidel v případě, že je detekována událost (log), který např. prokazuje, že došlo k úspěšné authentizaci nebo jiné události, která vypovídá o nikoli nelegitimní povaze předchozích detekovaných událostí. Tímto by se zabránilo situacím, ve kterých administrátor čtyřikrát zadá špatné authentizační údaje a poté správné, odhlásí se, bude se chtít přihlásit znovu, zadá však opět špatné údaje a bude vůči němu provedená akce (blacklist ve firewallu) a nebude se moci již dále přihlásit.

10. Závěr

Zadáním této práce bylo navrhnout a implementovat systém pro monitorování záznamů o transakcích služeb na OS Linux v reálném čase. V rámci této práce bylo dbáno na detailní analýzu již existujících řešení na trhu. Důvodem bylo zejména zamezení vyvíjení již notorietně známých věcí a postupů zcela od začátku. Tímto se práce mohla detailněji zaměřit na aspekty nové a v současných software chybějící. Dalším důležitým cílem práce bylo promyslet druhy útoků, které by řešení mohlo na konkrétních službách vymezených v zadání detekovat. Bylo vytvořeno několik pravidel na bázi regulárních výrazů pro detekci konkrétních událostí ve službách SSH, Apache, Postfix a Dovecot.

Práce, potažmo implementované řešení, je dílem, které sotva kdy dosáhne bodu, kde již na ní nebude co zlepšit. Nicméně, ve smyslu slov Senecy – „Umění je dlouhé, život krátký“[13][36] – je předkádaná práce výsledkem, který je velmi dobře použitelný pro nasazení v praxi. Ve smyslu interpretace tohoto aphorismu lze konkludovat, že práce předkládá rozumný výsledek, který je vystavěn na principech modularity, versatility, zásadách ochrany osobních údajů a které je jednoduché dále rozvíjet a udržovat. Bylo dosaženo všech cílů, které byly v zadání vymezeny.

Řešení bylo vyvíjeno na operačním systému Fedora 29 Workstation. Byla použita verze Python interpreta `3.6.8`, jež současně představuje verzi jedinou, na které bylo řešení testováno. Při vývoji řešení nastal problém zejména při implementací databáze. Pro efektivní práci byla snaha použít modul `dataset` pro ukládání dat do databáze. Modul `dataset` však neustále produkoval nepředvídatelné a stěží pochopitelné chyby, které se velmi obtížně ladily. Proto bylo nakonec ustoupeno od jeho použití a byl použit modul `sqlite3`. Toto řešení přidalo pár řádků kódu a pár hodin práce navíc, výsledek však stojí za to.

[13] Ars longa, vita brevis.

Seznam zkratek

CIDR	Classless Inter-Domain Routing
CLI	Command Line Interface
DNS	Domain Name System
DoS	Denial of Service
ELF	Extended Log Format
GDPR	General Data Protection Regulation
GGH	GoofyGoHome
HTTP	Hypertext Transfer Protocol
IP	Internet Protocol
JSON	JavaScript Object Notation
NCSA	National Center for Supercomputing Applications
OS	Operating System
Pf	Packet filter
PID	Process Identification number
SDEU	Soudní dvůr Evropské unie
SFEU	Smlouva o fungování Evropské unie
SMTP	Simple Mail Transfer Protocol
SOP	Same Origin Policy
TTY	Teletypewriter
W3C	World Wide Web Consortium
XML	Extensible Markup Language

Literární a jiné informační zdroje

[1] Brief introduction of log file formats. *Loganalyzer.net* [online]. 2014 [cit. 2018-12-12]. Dostupné z: https://www.loganalyzer.net/log-analysis/log-file-format.html

[2] CUPPENS, Frédéric, Nora CUPPENS-BOULAHIA, Joaquin GARCIA-ALFARO, Tarik MOATAZ, Stéphane MORUCCI a Xavier RIMASSON. *Detection des anomalies dans les pare-feux de nouvelles générations* [online]. Rennes, Francie: swid.fr [cit. 2018-11-21]. Dostupné z: https://pdfs.semanticscholar.org/e71a/27e475363aba38b2dedbea0c976babaa343c.pdf.

[3] Extended Log File Format. *W3C* [online]. [cit. 2018-12-12]. Dostupné z: https://www.w3.org/TR/WD-logfile.html.

[4] Fail2Ban. *Fail2Ban* [online]. 2016 [cit. 2018-12-12]. Dostupné z: https://www.fail2ban.org/wiki/index.php/Main_Page.

[5] FCNTL(2) – Linux Programmer's Manual: fcntl - manipulate file descriptor. *Man7.org* [online]. 2. 2. 2018 [cit. 2018-12-12]. Dostupné z: http://man7.org/linux/man-pages/man2/fcntl.2.html.

[6] GIFT, Noah a Jeremy M. JONES. *Python for Unix and Linux system administration*. Beijing: O'Reilly, 2008, 433 s. ISBN 978-0596515829.

[7] G. Yves. Pyruse. 2017. *GitHub repozitář* [online]. [cit. 2018-12-12]. Dostupné z: https://yalis.fr/git/yves/pyruse.

[8] G., Yves. Pyruse 1.0 : pour remplacer Fail2ban et autres « scruteurs » de journaux sur un GNU/Linux moderne. In: *Linuxfr.org* [online]. 154 rue de Picpus 75012 Paris - France: L'association LinuxFr, 2018, 12. 02. 18 [cit. 2018-12-12]. Dostupné z: https://linuxfr.org/news/pyruse-1-0-pour-remplacer-fail2ban-et-autres-scruteurs-de-journaux-sur-un-gnu-linux-moderne.

[9] HANSEN, Marit. Data Protection by Default in Identity-Related Applications. In: *Simone Fischer-Hübner, Elisabeth de Leeuw, Chris Mitchell (Eds.): Policies and Research in Identity Management*. Third IFIP WG 11.6 Working Conference (= IFIP Advances in Information and Communication Technology. Svazek 396). Springer, Heidelberg / Berlin 2013, ISBN 978-3-642-37281-0, DOI:10.1007/978-3-642-37282-72. Dostupné z: https://hal.inria.fr/hal-01470500/document.

[10] HOEREN, Thomas. *Internetrecht* [online]. April 2016. Münster: Institut für Informations-, Telekommunikations- und Medienrecht, Universität Münster, 2016 [cit. 2018-11-18]. Dostupné z: https://www.uni-muenster.de/Jura.itm/hoeren/materialien/Skript/Skript%20Internetrecht_April_2016.pdf.

[11] HOMRI, Raja. *Conception et Développement d'un Système d'analyse et Reporting des Fichiers Logs*. Mutuelleville, Tunis, 2016. Diplomová práce. UVT. Vedoucí práce Ahlem Ben Hassine. Dostupné z: http://pf-mh.uvt.rnu.tn/869/1/Conception-developpement-systeme-analyse-reporting-fichiers-logs.pdf.

[12] Ist IP Adresse ein personenbezogenes Datum? – der (vorerst) letzte Akt. In: *SüdWest Datenschutz* [online]. Karlsruhe, 25. 9. 2017 [cit. 2018-12-12]. Dostupné z: https://www.suedwest-datenschutz.com/sind-ip-adressen-personenbezogene-daten-der-vorerst-letzte-akt/.

[13] KEHR, Flavius, Daniel WENTZEL a Tobias KOWATSCH. Privacy Paradox Revised: Pre-Existing Attitudes, Psychological Ownership, and Actual Disclosure. In: *International Conference on Information Systems* 2014 [online]. Auckland, 2014 [cit. 2018-12-12]. Dostupné z: https://www.alexandria.unisg.ch/242216/1/PDF%20Proof_revised.pdf.

[14] KENT, Karen a Murugiah SOUPPAYA. *Guide to computer security log management: Recommendations of the National Institute of Standards and Technology* [online]. Gaithersburg, MD, USA: NIST, 2006 [cit. 2018-12-12]. Dostupné z: https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-92.pdf.

[15] Locking in the Linux Kernel. *Kernel.org* [online]. [cit. 2018-12-13]. Dostupné z: https://www.kernel.org/doc/htmldocs/kernel-locking/locks.html.

[16] Logging best practices. *Splunk.com* [online]. 2018 [cit. 2018-12-12]. Dostupné z: http://dev.splunk.com/view/logging/SP-CAAAFCK.

[17] LANDSTEINER, Norbert. *Mass:werk – media environments* [online]. 2003 [cit. 2019-05-22]. Dostupné z: https://www.masswerk.at/termlib/.

[18] Lua. *Lua* [online]. 2018 [cit. 2018-12-12]. Dostupné z: https://www.lua.org/about.html.

[19] MANDL, Peter. *Grundkurs Betriebssysteme*. Čtvrté vydání. Springer Vieweg, 2014. ISBN 978-3-658-06217-0.

[20] MASQUELIER, Gérald, Antoine MOTTIER a Cédric PRONZATO. *Les Firewalls* [online]. Paris, 2006 [cit. 2018-12-12]. Dostupné z: http://igm.univ-mlv.fr/~duris/NTREZO/20052006/MasquelierMottierPronzato-Firewall-rapport.pdf. Université Paris-Est Marne-la-Vallée.

[21] MÄCHTEL, Michael a Jürgen QUADE. *Moderne Realzeitsysteme kompakt: Eine Einführung mit Embedded Linux*. Heidelberg: dpunkt.verlag, 2012. ISBN 978-3-86491-217-7.

[22] MURÍNOVÁ, Júlia. *Application Log Analysis* [online]. Brno, 2015 [cit. 2018-12-12]. Dostupné z: https://is.muni.cz/auth/th/sw6lm/thesis_murinova.pdf. Diplomová práce. Masarykova univerzita.

[23] MYŠKA, Matěj, Radim POLČÁK, Jaromír ŠAVELKA, Libor KYNCL a Iveta SVIRÁKOVÁ. *Veřejné licence v České republice* [online]. Druhé vydání. Brno: Masarykova univerzita, 2014 [cit. 2018-11-11]. ISBN 978-80-210-7193-3. Dostupné z: https://is.muni.cz/repo/1203341/Myska_et_al._-_Verejne_licence_2.0_-_online.pdf.

[24] Nařízení Evropského parlamentu a Rady (EU) 2016/679 ze dne 27. 4. 2016 o ochraně fyzických osob v souvislosti se zpracováním osobních údajů a o volném pohybu těchto údajů a o zrušení směrnice 95/46/ES (obecné nařízení o ochraně osobních údajů). In: *EUR-Lex* [online právní informační systém]. Uřad pro publikace Evropské unie [cit. 4. 12. 2018]. Dostupné z: https://eur-lex.europa.eu/legal-content/CS/TXT/PDF/?uri=CELEX:32016R0679.

[25] Nález Ústavního soudu ze dne 22. 3. 2011, sp. zn. Pl. ÚS 24/10. In: *NALUS* [online]. [cit. 2018-12-12]. Dostupné z: http://nalus.usoud.cz/Search/GetText.aspx?sz=Pl-24-10_1.

[26] Německý Spolkový ústavní soud, rozsudek z 2. března 2010, 1 BvR 256/08. In: *Bundesverfassungsgericht* [online]. [cit. 2018-12-12]. Dostupné z: https://www.bundesverfassungsgericht.de/entscheidungen/rs20100302_1bvr025608.html.

[27] NULÍČEK, Michal et al. *GDPR - obecné nařízení o ochraně osobních údajů*. 2. vydání. Praha: Wolters Kluwer, 2018. Praktický komentář. ISBN 978-80-7598-068-7.

[28] PACHECO, Peter. *An Introduction to Parallel Programming*. San Francisco, CA, USA: Morgan Kaufmann Publishers, 2011. ISBN 978-0123742605.

[29] Positive security model. *OWASP* [online]. 2015 [cit. 2018-12-12]. Dostupné z: https://www.owasp.org/index.php/Positive_security_model.

[30] Python 2.7.15 documentation: mutex — Mutual exclusion support. *Python.org* [online]. [cit. 2018-12-12]. Dostupné z: https://docs.python.org/2/library/mutex.html.

[31] Rozhodnutí německého Spolkového soudního dvora ze dne 28. 10. 2014, sp. zn.: VI ZR 135/13. In: *Juris BGH*. Dostupné z: http://juris.bundesgerichtshof.de/cgi-bin/rechtsprechung/document.py?Gericht=bgh&Art=en&nr=69680&pos=0&anz=1.

[32] Rozsudek Soudního dvora (druhého senátu) ze dne 19. října 2016. Patrick Breyer proti Bundesrepublik Deutschland. Věc C-582/14. In: *Curia* [právní informační systém]. Soudní dvůr Evropské unie [cit. 5. 12. 2018]. Dostupné z:http://curia.europa.eu/juris/document/document.jsf?text=&docid=184668&pageIndex=0&doclang=cs&mode=lst&dir=&occ=first&part=1&cid=1403270.

[33] Rozsudek Soudního dvora (velkého senátu) ze dne 10. února 2009. Věc C-301/06, Irsko proti EP/Radě. In: *Curia* [právní informační systém], 2018 [cit. 18. 11. 2018]. Dostupné z: http://curia.europa.eu/juris/document/

document.jsf?text=&docid=72843&pageIndex=0&doclang=DE&mode=req&dir=&occ=first&part=1.

[34] RF-232. *Serveur SME-9.x/8.x & Fail2ban* [online]. Montréal, Québec, 2016 [cit. 2019-05-21]. Dostupné z: https://www.micronator.org/PDF/SME/SME-9.1_Fail2ban/RF-232_SME-9.1_fail2ban.pdf.

[35] SALMON, N. *Protéger son serveur avec Fail2ban* [online]. 2016 [cit. 2019-05-21]. Dostupné z: http://idum.fr/spip.php?page=spipdf&spipdf=pdf_article&id_article=303&nom_fichier=article_303.

[36] SENECA, L. Annaeus. *De brevitate vitae: Lateinisch/Deutsch, Von der Kürze des Lebens*. Stuttgart: Reclam, 2008. ISBN 978-3-15-018545-2.

[37] SPENNEBERG, Ralf. *Linux-Firewalls mit iptables & Co: Sicherheit mit Kernel 2.4 und 2.6 für Linux-Server und -Netzwerke*. Pearson Deutschland, 2006, 641 s. ISBN 9783827321367.

[38] SSH Authentication Methods. *SecureBlackBox* [online]. /n software, 2019 [cit. 2019-05-22]. Dostupné z: https://www.secureblackbox.com/kb/articles/SSH-Authentication-methods.rst.

[39] VALENTA, Ondřej. *Definice transformačních pravidel pro logovací data z distribucí Debian a Ubuntu* [online]. Brno, 2016 [cit. 2018-12-12]. Dostupné z: https://is.muni.cz/auth/th/lytt9/BAKALARSKA_PRACE_hotovo.pdf. Bakalářská práce. Masarykova univerzita.

Přílohy

A. Obsah CD – praktická část práce

Součástí této práce je praktické řešení, které je ve své úplnosti uloženo na optickém médiu, jež je k práci přiloženo. Toto médium obsahuje veškeré materiály potřebné pro úspěšné ověření funkčnosti naprogramovaného řešení. Přiložené optické médium kromě zdrojových kódů a několika pomocných skriptů obsahuje také elektronickou verzi této klasifikační práce.

Jednotlivé soubory a adresáře jsou uvedeny v souladu s jejich hierarchickou strukturou, ve které jsou uloženy na přiloženém médiu. Přiložené CD obsahuje následující adresáře a soubory:

- `bp-xhoder01.pdf`

  Teoretická část bakalářské klasifikační práce.

- `goofygohome/`

  Adresář obsahující veškeré soubory se zdrojovými kody implementovaného řešení.

- `goofygohome/ggh_daemon.py`

  Soubor zdrojového kodu obsahující především funkce zpracování argumentů, kontroly jediné instance, vytvoření PID souboru a uvedení do modu démona. Je nutné jej zpravidla spouštět v režimu správce, a to z důvodu redelegace těchto práv na příkazy (hodnoty X_ACTION v konfiguračním soufboru) tímto skriptem spouštěných. Toto však neplatí, pokud hodnoty X_ACTION nevyžadují zvýšená privilegia, což ale bývá z podstaty věci spíše výjimkou.

- `goofygohome/comm_server.py`

  Soubor zdrojového kódu, kde jsou obsaženy funkce pro zajištění komunikace na bázi lokálního linuxového socketu.

- `goofygohome/cmd_processor.py`

  Soubor zdrojového kódu zpracovávající příkazy zaslané klienty, kterými se démon ovládá (`ggh_control_cli.py nebo ggh_control_web.py`).

- `goofygohome/ggh_control_cli.py`

Soubor zdrojového kódu klienta CLI, který umožňuje ovládání démona na lokální úrovni přes rozhraní CLI.

- `goofygohome/ggh_control_web.py`

  Soubor zdrojového kódu klienta, který umožňuje ovládání démona vzdáleně skrz protokol HTTP, tedy přes webový prohlížeč.

- `goofygohome/config.py`

  Soubor zdrojového kódu obsahující definice tříd opatřující parsování konfiguračního souboru `conf.conf`. Nastavení obsažené v `conf.conf` se přenese do datové struktury, se kterou program dále pohodlně pracuje a ze které lze k nastavení pro jednotlivé služby snadno přistupovat.

- `goofygohome/constants.py`

  Soubor zdrojového kódu, kde jsou definovány konstantní proměnné. V současné podobě práce se jedná toliko o proměnnou vypovídající o verzi GGH.

- `goofygohome/config_validity_checker.py`

  Soubor zdrojového kódu, který obsahuje třídu starající se o kontrolu správnosti nastavení konfigurace, pokud je změna konfigurace činěna prostřednictvím webového klienta.

- `goofygohome/rule_executor.py`

  Soubor zdrojového kódu, ve kterém je algoritmus pro kontrolu journal souborů a systému journald.

- `goofygohome/run_command.py`

  Soubor zdrojového kódu převzatý v rámci citační zákonné licence z Fail2Ban. Obsahuje definice funkcí v *lato sensu* pro spuštění systémových příkazů – spouštění hodnot atributů `ACTION` a `ANTIACTION`.

- `goofygohome/filetracker.py`

  Soubor zdrojového kódu převzatý z DenyHosts a následně modifikovaný, který má funkci kontrolovat, zdali nastala změna v journalovém souboru, a starat se o uchovávaní informací o posledním zkontrolovaném logu i mezi restarty GGH.

- `goofygohome/database.py`

  Soubor zdrojového kódu obsahující třídu Database, která zajišťuje zápis a čtení dat do databázového souboru v adresáři `data/`.

- `goofygohome/Makefile`

  Soubor make sloužící současně pouze k automatizovanému mazání dočasných souborů `*.pyc`. K tomuto stačí zadat do příkazové řádky příkaz `make clean` v předmětném adresáři.

- `goofygohome/conf/`

Adresář obsahující konfigurační soubor démona, kde se konfigurují pravidla detekce (`conf.conf`) a konfigurační soubor, ve kterém se definují authentizační údaje pro webové rozhraní (klienta).

- `goofygohome/conf/conf.conf`

  Konfigurační soubor umožňující konfiguraci všeobecných nastavení programu a nastavení pro pravidla detekce pokusů o narušení bezpečnosti jednotlivých definovaných služeb.

- `goofygohome/conf/conf.conf`

  Konfigurační soubor umožňující konfiguraci authentizačních údajů pro webové ovládací prostředí.

- `goofygohome/data/`

  Adresář, do kterého se ukádají interní datové soubory vygenerované démonem a sloužící jeho správné funci. Adresář obsahuje soubor databáze, soubor journal, soubor pid, kam se ukládá PID procesu démona, a nakonec soubory s uloženou poslední pozicí čtení souborů journal, které démon hlídá.

- `goofygohome/html/`

  Tento adresář obsahuje soubory HTML, JS, CSS a další, které potřebuje `ggh_control_web.py` ke správnému zobrazení a funkci webového prostředí.

- `goofygohome/webserver_data/`

  Adresář obsahující soubory vytvořené za běhu webového serveru (klienta pro ovládání démona).

- `goofygohome/LICENCE`

  Soubor obsahující veřejnou licenční smlouvu GPLv2, která je licenční smlouvou ve smyslu § 46 a násl. zákona č. 121/2000 Sb., autorský zákon, ve znění pozdějších předpisů, která, je–li přijata, konstituuje právo tento software užít za podmínek v licenční smlouvě stanovených. Poskytnutí tohoto software licenční smlouvou GPLv2, pod kterou je tento software dán k dispozici blíže nespecifikovanému a neomezenému okruhu subjektů, je dostáno právním závazkům plynoucích z užití částí počítačových programů v tomto software zveřejněných pod stejnou či jinou kompatibilní licencí. Byl-li v práci použit software standardně dostupný pod veřejnou licencí s GPLv2 nekompatibilní, nebyla taková licence přijata, nýbrž byla daná část předmětného software užita v rámci licence zákonné (§ 31 odst. 1 písm. a) zákona č. 121/2000 Sb., autorský zákon).

B. Návod pro spuštění řešení v základní konfiguraci a pro demonstrativní účely

Předpokládá se pracovní adresář nastaven na adresář `goofygohome` v podobě, v jaké je uložen na přiloženém médiu.

V konfiguračním souboru `conf/conf.conf` v základní podobě se nacházejí předdefinovaná ilustrativní pravidla detekce, které slouží primárně jako návod pro vytváření pravidel vlastních.

Pro demonstrativní účely je stěžejní zejména služba s názvem `[test]`, která obsahuje dvě pravidla detekce – `test_rule1` a `test_rule2`. Služba `[test]` definuje `LOG_LOCATION` jako cestu k journal souboru `misc/mail.log`[14], který se nachází v indikované lokaci. Jedná se o journal soubor naplněný především logy služby postfix. Jsou z něj zřejmé i četné selhané pokusy o authentizaci. Dále je v definici této služby nastaven parametr `RETROACTIVE` na hodnotu `true`, což znamená, že journal soubor bude prohledáván za účelem nalezení shody s pravidly `test_rule1` a `test_rule2` i zpětně. Soubor bude zkontrolován od začátku, i když dané události v něm zaznamenané nastaly dříve, než byl spuštěn samotný program (řešení).

Pro spuštění řešení v režimu démona je potřeba do příkazové řádky zadat `./ggh_daemon.py --start`. Pro spuštění v režimu `debug` je nutno na konec příkazu přidat parametr `-d`. Démona lze takto spustit pouze pokud mu jsou přidělena náležitá práva. Direktiva[15] na začátku souboru zajistí, že pro interpretaci zdrojového kódu bude použit python verze 3. Po spuštění program zpracuje konfigurace a definice pravidel v konfiguračním souboru a začne s kontrolou. Jelikož je služba `[test]` nastavena jako retroaktivní, začne program s prohledáváním od daného journal souboru od začátku, tedy aniž by od doby spuštění nastala jakákoli log událost ve kterékoli ze sledovaných služeb.

Pro spuštění webového klienta, tedy webového prostředí pro ovládání démona přes HTTP, se zadá do příkazové řádky příkaz `./ggh_control_web.py`, opět s parametrem `-d`, pokud je žádoucí debug mód. Webové rozhraní je poté přístupné na adrese `http://localhost:8008/`. Klienta CLI lze spustit obdobně příkazem `./ggh_control_cli.py`.

Zastavit démona lze příkazem `./ggh_daemon.py --stop`. Webový server lze zastavit příkazem `./ggh_control_web.py --stop`.

C. Pomocné skripty

- `goofygohome/kill.sh`

  Shell skript pro násilné ukončení běžícího webového serveru nebo démona. Pokud je skript spuštěn bez argumentu, je standardně ukončen démon. Pokud je jako argument skriptu `web`, je ukončen webový server pro ovládání přes HTTP. Pokud byl program, jenž má být tímto skriptem ukončen, spuštěn v režimu zvýšených privilegií, musí rovněž tento skript být spuštěn v režimu dostatečných privilegií.

[14] Soubor misc/mail.log je journal souborem s logy služeb postfix/smtpd, pop3d, imapd a dalších, který byl při vývoji používán pro testování pravidel. Soubor netvoří součást odevzdávané práce z důvodu obsahu osobních údajů.

[15] Jedná se o direktivu #!/usr/bin/env python3.

D. Defaultní konfigurační soubor

```
1  ########################################
2  #### GoofyGoHome configuration file ####
3  ########################################

5  ##################################################################
6  # Obligatory general settings:
7  # -> SOCKET_PATH -> always in section [general]
8  # -> DAEMON_SLEEP -> always in section [general]
9  # -> DB_EVENT_CHECK_SLEEP -> always in section [general]
10 #
11 # Facultative general settings:
12 # -> JAILTIME -> can be in [general] or in a particular rule
13 # -> THRESHOLDCOUNT -> can be in [general] or in a particular rule
14 # -> HOSTS_DENY -> can be only in [general]
15 # -> HOSTS_ALLOW -> can be only in [general]
16 # -> DO_NOT_DUPLICATE -> can be only in [general]
17 #
18 # Obligatory rule settings:
19 # -> LOG_LOCATION -> must be in rule section
20 # -> RULENAME -> must be in rule section
21 # -> REGEX -> must be in rule section
22 #
23 # Facultative rule setting:
24 # -> RETROACTIVE -> can be only in rule
25 # -> CRITERIA_TO_DISTINGUISH -> can be only in rule
26 # -> THRESHOLDCOUNT -> can be in [general] or in a particular rule
27 # -> ACTION -> can be only in rule
28 # -> ANTIACTION -> can be only in rule
29 # -> JAILTIME -> can be in [general] or in a particular rule
30 ##################################################################

32 [general]

34 # Path to socket file used for communication with cli/web client
35 SOCKET_PATH = /tmp/GGH-control-socket

37 # HOSTS_ALLOW is privileged to HOSTS_DENY
38 HOSTS_DENY = 10.75.5.0/24
39 HOSTS_ALLOW = 185.234.219.0/24,192.168.0.0/16

41 # Interval between journal file checks for new events
42 DAEMON_SLEEP = 30s
43 # Interval between database events checks
44 DB_EVENT_CHECK_SLEEP = 35s

46 # purge entries in DB that are older than 1 week
47 # 0 for disable
48 JAILTIME = 1w
```

```
50 THRESHOLDCOUNT = 10

52 DO_NOT_DUPLICATE = true

54 ###################################################################
55 ###################################################################
56 #### To check iptables INPUT rules:
57 #### sudo iptables -L INPUT -v -n | more
58 #### To ban a certain IP in iptables:
59 #### iptables -I INPUT -s <adresseIP_ggf> -p tcp -j DROP
60 ###################################################################
61 ###################################################################

63 [test]
64 # Sample logfile with pre-generated logs for testing purposes
65 LOG_LOCATION = misc/mail.log

67 # Default value is false
68 RETROACTIVE = true

70 # In context of every service the rule-IDs must differ

72 1_RULENAME = test_rule1
73 1_ENABLED = true
74 1_CRITERIA_TO_DISTINGUISH = adresseIP_ggf
75 1_REGEX = ".*: warning: unknown\[(?P<adresseIP_ggf>.*?)\]: SASL
       ↪ LOGIN authentication failed: authentication failure"
76 1_THRESHOLDCOUNT = 5
77 1_ACTION = "iptables -I INPUT -s adresseIP_ggf -p tcp --destination-
       ↪ port 25 -j DROP"
78 1_JAILTIME = 1d
79 1_ANTIACTION = "iptables -D INPUT -s adresseIP_ggf -p tcp --
       ↪ destination-port 25 -j DROP"

81 2_RULENAME = test_rule2
82 2_ENABLED = true
83 2_CRITERIA_TO_DISTINGUISH = adresseIP_ggf
84 2_REGEX = "^Failed password for (?P<user_ggf>.*) from (?P<
       ↪ adresseIP_ggf>[^ ]*) port"
85 2_THRESHOLDCOUNT = 3
86 2_ACTION = "echo action launched"
87 2_JAILTIME = 1d
88 2_ANTIACTION = "echo action undone"


91 [sshd]
92 LOG_LOCATION = journald

94 RETROACTIVE = false
```

```
96  # Detects failed authentication attempts regardless what auth.
        ↪ method has been used
97  3_RULENAME = x_ssh_failed_login1
98  3_ENABLED = true
99  3_CRITERIA_TO_DISTINGUISH = adresseIP_ggf
100 3_REGEX = "Failed (?P<method>\S*) for (?P<invalid>invalid user |
        ↪ illegal user )?(?P<user>.*) from (::ffff:)?(?P<adresseIP_ggf
        ↪ >[^ ]*)( port \d+)?( ssh2)?$"
101 3_THRESHOLDCOUNT = 5
102 3_ACTION = "iptables -I INPUT -s adresseIP_ggf -p tcp --destination-
        ↪ port 22 -j DROP"
103 # Values JAILTIME and ANTIACTION are not obligatory
104 3_JAILTIME = 1d
105 3_ANTIACTION = "iptables -D INPUT -s adresseIP_ggf -p tcp --
        ↪ destination-port 22 -j DROP"

107 # This rule is restricted to detect solely the "password" ssh method
108 4_RULENAME = xx_ssh_failed_login2
109 4_ENABLED = true
110 4_CRITERIA_TO_DISTINGUISH = adresseIP_ggf
111 4_REGEX = "^Failed password for (?P<user>.*) from (?P<adresseIP_ggf
        ↪ >[^ ]*) port"
112 4_THRESHOLDCOUNT = 5
113 4_ACTION = "iptables -I INPUT -s adresseIP_ggf -p tcp --destination-
        ↪ port 22 -j DROP"
114 4_JAILTIME = 1d
115 4_ANTIACTION = "iptables -D INPUT -s adresseIP_ggf -p tcp --
        ↪ destination-port 22 -j DROP"



118 # Apache -> the name of the service has to correspond to the name of
        ↪  systemd service name (in Fedora 29 apache2 is run as httpd)
119 [httpd]
120 LOG_LOCATION = /var/log/httpd/access_log
121 RETROACTIVE = false

123 100_RULENAME = apache_statuscode
124 100_ENABLED = true
125 100_CRITERIA_TO_DISTINGUISH = adresseIP_ggf
126 100_REGEX = "^(?P<adresseIP_ggf>[^\s]+) .* "(GET|POST|OPTIONS) (?P<
        ↪ url>.*) HTTP/[012].[012]" (?P<statuscode>[4-5]\d\d)"
127 100_THRESHOLDCOUNT = 5
128 100_ACTION = "iptables -I INPUT -s adresseIP_ggf -p tcp --
        ↪ destination-port 80 -j DROP"
129 100_ANTIACTION = "iptables -D INPUT -s adresseIP_ggf -p tcp --
        ↪ destination-port 80 -j DROP"
130 100_INTERVAL = 1d

132 200_RULENAME = apache_http_auth
133 200_ENABLED = true
134 200_CRITERIA_TO_DISTINGUISH = adresseIP_ggf
```

```
200_REGEX = "^(?P<adresseIP_ggf>[^\s]+) .* "(GET|POST|OPTIONS) (?P<
    ↪ url>.*) HTTP/[012].[012]" 401"
200_THRESHOLDCOUNT = 5
200_ACTION = "iptables -I INPUT -s adresseIP_ggf -p tcp --
    ↪ destination-port 80 -j DROP"
200_ANTIACTION = "iptables -D INPUT -s adresseIP_ggf -p tcp --
    ↪ destination-port 80 -j DROP"
200_INTERVAL = 1d


[dovecot]

LOG_LOCATION = journald
RETROACTIVE = false

1_RULENAME = dovecot_auth
1_ENABLED = true
1_CRITERIA_TO_DISTINGUISH = adresseIP_ggf
1_REGEX = ".* dovecot: pop3-login: Aborted login: user=, method=
    ↪ PLAIN, rip=::ffff:\[(?P<adresseIP_ggf>.*?)\],"
1_THRESHOLDCOUNT = 5
1_ACTION = "iptables -I INPUT -s adresseIP_ggf -p tcp -j DROP"
1_JAILTIME = 1d
1_ANTIACTION = "iptables -D INPUT -s adresseIP_ggf -p tcp -j DROP"


[postfix]

LOG_LOCATION = journald
RETROACTIVE = false

1_RULENAME = postfix_auth
1_ENABLED = true
1_CRITERIA_TO_DISTINGUISH = adresseIP_ggf
1_REGEX = ".*: warning: unknown\[(?P<adresseIP_ggf>.*?)\]: SASL
    ↪ LOGIN authentication failed: authentication failure"
1_THRESHOLDCOUNT = 5
1_ACTION = "iptables -I INPUT -s adresseIP_ggf -p tcp -j DROP"
1_JAILTIME = 1d
1_ANTIACTION = "iptables -D INPUT -s adresseIP_ggf -p tcp -j DROP"
```